A Computational Theory of Metaphor

James H. Martin

Copyright © 1988

Report Documentation Page

Form Approved
OMB No. 0704-0188

Public reporting burden for the collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to a penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number.

1. REPORT DATE **NOV 1988**	2. REPORT TYPE	3. DATES COVERED **00-00-1988 to 00-00-1988**
4. TITLE AND SUBTITLE **A Computational Theory of Metaphor**		5a. CONTRACT NUMBER
		5b. GRANT NUMBER
		5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)		5d. PROJECT NUMBER
		5e. TASK NUMBER
		5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **University of California at Berkeley,Department of Electrical Engineering and Computer Sciences,Berkeley,CA,94720**		8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)		10. SPONSOR/MONITOR’S ACRONYM(S)
		11. SPONSOR/MONITOR’S REPORT NUMBER(S)
12. DISTRIBUTION/AVAILABILITY STATEMENT **Approved for public release; distribution unlimited**		
13. SUPPLEMENTARY NOTES		

14. ABSTRACT

Metaphor is a conventional and ordinary part of language. A theory attempting to explain metaphor must account for the ease with which conventional metaphors are understood, and with the ability to understand novel metaphors as they are encountered. An approach to metaphor, based on the explicit representation of knowledge about metaphors, has been developed to address these issues. This approach asserts that the interpretation of conventional metaphoric language should proceed through the direct application of specific knowledge about the metaphors in the language. Correspondingly, the interpretation of novel metaphors can be accomplished through the systematic extension, elaboration, and combination of knowledge about already well-understood metaphors. MIDAS (Metaphor Interpretation, Denotation and Acquisition System) is a computer program that embodies this approach. MIDAS can be used to perform the following tasks: represent knowledge about conventional metaphors, interpret metaphoric language by applying this knowledge, and dynamically learn new metaphors as they are encountered during normal processing. Knowledge about conventional metaphors is represented in the form of coherent sets of associations between disparate conceptual domains. The representation captures both the details of individual metaphors, and the systematicities exhibited by the set of metaphors in the language as a whole. These systematic sets of associations were implemented using the KODIAK knowledge representation language. MIDAS is capable of using this metaphoric knowledge to interpret conventional metaphoric language. The main thrust of this approach is that normal processing of metaphoric language proceeds through the direct application of specific knowledge about the metaphors in the language. This approach gives equal status to all conventional metaphoric and literal interpretations. Moreover, the mechanisms used to arrive at metaphoric and literal interpretations are fundamentally the same. When a metaphor is encountered for which MIDAS has no applicable knowledge, MIDAS calls upon its learning component - the Metaphor Extension System (MES). The approach embodied in the MES asserts that a novel metaphor can best be understood through the systematic extension of an already well-understood metaphor. MIDAS has been integrated as a part of the UNIX Consultant system. UC is a natural language consultant system that provides naive computer users with advice on how to use the UNIX operating system. By calling upon MIDAS, UC can successfully interpret and learn conventional UNIX domain metaphors, as they are encountered during the course of UC's normal processing.

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**	**Same as Report (SAR)**	**224**	

A Computational Theory of Metaphor

James H. Martin

ABSTRACT

Metaphor is a conventional and ordinary part of language. A theory attempting to explain metaphor must account for the ease with which conventional metaphors are understood, and with the ability to understand novel metaphors as they are encountered. An approach to metaphor, based on the explicit representation of knowledge about metaphors, has been developed to address these issues. This approach asserts that the interpretation of conventional metaphoric language should proceed through the direct application of specific knowledge about the metaphors in the language. Correspondingly, the interpretation of novel metaphors can be accomplished through the systematic extension, elaboration, and combination of knowledge about already well-understood metaphors.

MIDAS (Metaphor Interpretation, Denotation, and Acquisition System) is a computer program that embodies this approach. MIDAS can be used to perform the following tasks: represent knowledge about conventional metaphors, interpret metaphoric language by applying this knowledge, and dynamically learn new metaphors as they are encountered during normal processing.

Knowledge about conventional metaphors is represented in the form of coherent sets of associations between disparate conceptual domains. The representation captures both the details of individual metaphors, and the systematicities exhibited by the set of metaphors in the language as a whole. These systematic sets of associations were implemented using the KODIAK knowledge representation language.

MIDAS is capable of using this metaphoric knowledge to interpret conventional metaphoric language. The main thrust of this approach is that normal processing of metaphoric language proceeds through the direct application of specific knowledge about the metaphors in the language. This approach gives equal status to all conventional metaphoric and literal interpretations. Moreover, the mechanisms used to arrive at metaphoric and literal interpretations are fundamentally the same.

When a metaphor is encountered for which MIDAS has no applicable knowledge, MIDAS calls upon its learning component - the Metaphor Extension System (MES). The approach embodied in the MES asserts that a novel metaphor can best be understood through the systematic extension of an already well-understood metaphor.

MIDAS has been integrated as a part of the UNIX Consultant system. UC is a natural language consultant system that provides naive computer users with advice on how to use the UNIX operating system. By calling upon MIDAS, UC can successfully interpret and learn conventional UNIX domain metaphors, as they are encountered during the course of UC's normal processing.

Acknowledgements

I'd like to thank my advisor, Robert Wilensky, who diligently read all the drafts of this thesis. My long discussions with him resulted in many of the ideas in this thesis. I'd also like thank the other members of my thesis committee: Lotfi Zadeh and Paul Kay.

Berkeley is an amazing place to do language research. This is due in large part to the presence of Charles Fillmore, Paul Kay, and George Lakoff. I'd like to thank George, in particular, for opening the door for this kind of research.

Richard Alterman's unbridled enthusiasm and interest in my research came at a critical time in my graduate career. Peter Norvig put up with my incessant questions about how to do research and how to write a thesis. I'd like to thank Paul Jacobs for being the first one to show that it really could be done. Marc Luria and Jim Mayfield shared the various horrors of prelims, quals, and learning to do research. They were good companions on this long strange trip.

Over the last few years my office-mates have certainly made 545 Evans an interesting place to be. I'd like to thank Michael Braverman for all the stories that I thought would never end. Michael also served as an excellent reference for the latest in Machine Learning. The effects of Dan Jurafsky's seemingly endless supply of energy and enthusiasm cannot be overestimated. He also patiently answered all my naive linguistic questions. I'd like to thank Nigel Ward for the tuna and Dekai Wu for the daily fashion show. Over the years the following members of the Berkeley AI research group contributed to the ideas in this thesis, and were fun to have around: Yigal Arens, Joe Falletti, David Chin, Lisa Rau, Margaret Butler, Charles Cox, Terry Regier, Eric Karlson, and Anthony Albert. Sharon Tague has provided invaluable administrative support, in addition to keeping things interesting.

Quite a number of people contributed to the length, and occasionally to the quality, of my stay in Berkeley. I'd like to thank (in order of appearance): James Campbell, Tim Learmont, Margaret Butler, Yvonne Freund, and Aimee Fishkind. Leaving Berkeley would have been much harder if these people hadn't all fled first. Sandy Irani provided a welcome and necessary distraction over the last year while I wrote this beast. I'd also like to thank Andrew Romanowski and Cecelia Buchanan for their long-distance and long-term support.

My parents gave me their complete love and support throughout my graduate career. I can't possibly thank them enough for all they've done for me.

Finally I'd like to thank the taxpayers for providing the support for this work, which was administered through: the Defense Advanced Research Projects Agency (DoD), monitored by the Space and Naval Warfare Systems Command under Contracts N00039-84-C-0089 and N00039-88-C-0292; the Office of Naval Research, under grant N00014-80-C-0732. Additional support was provided by a GTE Laboratories Fellowship.

Contents

Chapter 1

Introduction

1.1. Conventional Metaphor

This thesis addresses the problem of understanding conventional metaphoric language. Consider the following examples.

(1) How can I *kill* a process?
(2) How can I *get into* Lisp?
(3) You can *enter* Emacs by typing 'emacs' to the shell.
(4) Nili *gave* Marc *her* cold.
(5) Inflation is *eating up* our savings.

The italicized words in each of these examples are being used to metaphorically refer to concepts that are quite distinct from those that might be considered the normal meanings of the words. Consider the use of *enter* in (3). *Enter* is being used, in this example, to refer to the actions on a computer system that result in the activation of a program. This use is clearly different from what might be called the ordinary or basic meaning of the word that has to do with the actions that result in an agent entering an enclosure.

While the word *enter* is used metaphorically in (3), this metaphor is neither novel nor poetic. Instead, the metaphorical use of *enter* results from a conventional systematic conceptual metaphor that allows computer processes to be viewed as enclosures. The various actions and states that have to do with containment are used to refer to actions and states that have to do with the activation, deactivation, and use of these computer processes. This conceptual metaphor, structuring processes as enclosures, underlies the normal conventional way of speaking about these processes. Therefore, the uses of the words *enter* in (3) and *get into* in (2) are ordinary conventional ways of expressing these concepts that nevertheless involve a metaphor.

1.2. The Metaphoric Knowledge Approach

The main thrust of my approach to metaphor is that the interpretation of conventional metaphoric language proceeds through the direct application of specific knowledge about the metaphors in the language. The interpretation of novel metaphors is accomplished through the systematic extension, elaboration, and combination of already well-understood metaphors. The proper way to approach the topic of metaphor, therefore, is to study the details of both individual metaphors and the system of metaphors in the language.

It is useful here to consider an analogy between the study of metaphor and the study of syntax. Broadly speaking, the study of syntax is concerned with the representation, use and acquisition of sets of complex facts that may be said to represent the grammar of a language. The approach taken in this thesis approaches the study of metaphor in a similar fashion. In particular, it addresses the *representation*, *use*, and *acquisition* of knowledge about the metaphors in the language.

This approach has been embodied in MIDAS (Metaphor Interpretation, Denotation, and Acquisition System). MIDAS is a set of computer programs that can be used to perform the following tasks: explicitly represent knowledge about conventional metaphors, use this knowledge to interpret metaphoric language, and learn new metaphors as they are encountered.

In order to make the problem of understanding metaphors more concrete, consider the implications of (1) through (3) for a system like the UNIX Consultant (Wilensky 1986). UC is a natural language consultant system that provides naive computer users with advice on how to use the UNIX operating system. Metaphors like those shown above are ubiquitous in technical domains like UNIX. A system that is going to accept natural language input from users and provide appropriate natural language advice must be prepared to handle such metaphorical language. Consider the following UC session illustrating the processing of (2).

```
> (do-sentence)
Interpreting sentence:

How can I get into lisp?

  Applying conventional metaphor Enter-Process.

UC: You can get into lisp by typing lisp to the shell.
```

In this example, the user has employed the conventional metaphor, described above, that entails that programs can be viewed as enclosures or environments. The action of entering such an enclosure, underlying the phrase *get into*, corresponds to the action that begins the use of the program. In order to appropriately handle this example, UC must be able to access and apply specific knowledge about this conventional metaphor.

UC handles this kind of metaphoric language by calling upon MIDAS. In this example, UC calls upon MIDAS to find a coherent interpretation for this use of *get into*. MIDAS finds and applies the conventional metaphor that allows the invocation of a program to be viewed as an entering. Chapter 5 gives the full details of how MIDAS interprets conventional metaphors for which it has adequate knowledge.

The previous example illustrates the situation where a user employs a conventional metaphor of which MIDAS has explicit knowledge. However, a system like MIDAS will inevitably encounter metaphors for which it has no adequate conventional knowledge. Consider the following UC session illustrating (1).

```
> (do-sentence)

Interpreting sentence:

How can I kill a process?

  No valid interpretations of killing.

=============================================================
Entering Metaphor Extension System
=============================================================

  Attempting to extend existing metaphor.
  Extending similar metaphor Kill-Conversation.
  Creating new metaphor: Killing-Terminate-Computer-Process

UC: You can  kill a computer process by typing ^c to the shell.
```

In this example, the user has employed the conventional UNIX metaphor that the termination of an ongoing process can be viewed as a killing. However, unlike in the previous example, MIDAS finds that it is initially unable to interpret this example because it has no knowledge of this conventional metaphor. This example illustrates the operation of the learning component of MIDAS, the Metaphor Extension System (MES). This system is invoked by MIDAS when it discovers a metaphor for which it has no adequate knowledge. The task of the MES is to attempt to extend its knowledge of some existing metaphor in a way that will yield a coherent interpretation for the new use and provide a basis for directly understanding similar uses in future. In this case, the system finds and extends a known metaphor that also uses *kill* to mean terminate. The full details of the MES component of MIDAS will be given in Chapters 6 through 9.

The approach to handling metaphor embodied in MIDAS is a reaction against the strategies employed in previous computational approaches to metaphor. Chapter 2 will provide a detailed analysis of these approaches. This section will serve to highlight the major differences between the approach taken here and these previous approaches.

Previous computational approaches to metaphor have adopted, what I call, a knowledge-deficient approach. By this, I mean an approach that makes no explicit use of knowledge about the metaphors in the language. The knowledge-deficient approach has been manifested in two distinct processing strategies.

The *word-sense* strategy (Riesbeck 1978, Wilensky and Arens 1980, Small 1982, Hirst 1983) recognizes that there are conventional uses of words that deviate from ordinary compositional, or literal, meaning. The word-sense strategy addresses this problem by merely listing each separate use as an isolated and unmotivated word-sense in the lexicon. While this approach adequately allows known conventional senses to be interpreted correctly, it nevertheless has a number of shortcomings.

The first shortcoming involves a representational issue. The listing of each separate use as an individual fact in the lexicon fails to capture the systematicities among senses of different words or among the senses of a single word. This enumeration of each use as an isolated and unmotivated fact about the language leads to the second shortcoming. The lack of structure in the knowledge makes it difficult to predict or classify the meaning of new uses when they are encountered. The approach taken in this thesis is an attempt to list these uses as conventional parts of our language knowledge while still capturing the rich conceptual structure that gives rise to them.

The other major paradigm (Wilks 1973, 1975, 1978, Russell 1974, Carbonell 1981, Dejong and Waltz 1983, Fass 1983, 1987) has held that metaphorical language is not a conventional part of the language and hence there can be no explicit knowledge of metaphor. Metaphors must therefore be treated as anomalous or ill-formed deviations from normal language. The basic approach to interpreting these metaphors has been to first recognize their presence by virtue of the fact that they deviate from the known semantics of a literal interpretation. The task of interpreting this anomalous input is then seen as a special purpose task requiring access to knowledge and inference techniques that are not a part of the normal language processing faculties.

The approach taken here is a reaction against these knowledge-deficient approaches. The metaphors in the language are viewed as conventional parts of the language. They are as conventional a part of the language as the rules of the grammar or words in the lexicon. The proper approach, therefore, is to study them from the perspective of normal language processing through the correct application of specific knowledge. Unlike the word-sense approach, this knowledge must reflect the rich structure underlying the conventional metaphors in the language and attempt to account for how these metaphors can be acquired.

1.3. Details of the Approach

This section will provide an overview of the three-part MIDAS approach to metaphor. In particular, it will give detailed examples of the following topics:

- **Representation:** The explicit representation in a knowledge-base of the conventional metaphors in the language in the form of explicit associations between concepts.
- **Interpretation:** The correct and efficient application of the above metaphoric knowledge to the interpretation of metaphoric language.
- **Learning:** The acquisition of new metaphors when examples are encountered for which no known metaphor provides a coherent explanation.

1.3.1. Representation

Consider the following example of a conventional UNIX metaphor. The metaphorical use of the word *in* reflects a systematic metaphorical structuring of processes as enclosures.

(6) I am *in* Emacs.

Metaphors like this may be said to consist of two sets of component concepts, a *source* component and a *target* component. The target consists of the concepts to which the words are actually referring. The source refers to the concepts in terms of which the intended target concepts are being viewed. In this example, the target concepts are those representing the state of currently using a computer process. The target concepts are those that involve the state of being contained within some enclosure.

The approach taken here is to explicitly represent conventional metaphors as sets of associations between source and target concepts. The metaphor specifies how the source concepts reflected in the surface language correspond to various target concepts. In this case, the metaphor consists of component associations that specify that the state of being enclosed represents the idea of currently using the editor; where the user plays the role of the enclosed thing, and the Emacs process plays the role of the enclosure.

These sets of metaphoric associations, along with the concepts that comprise the source and target domains, are represented using the KODIAK (Wilensky 1986) representation language. KODIAK is an extended semantic network language in the tradition of KL-ONE and its variants. The details of KODIAK and the representation of metaphoric knowledge will be fully described in Chapter 4.

These sets of metaphoric associations representing conventional metaphors in the language are full-fledged KODIAK concepts. As, such they can be related to other concepts and arranged in abstraction hierarchies using the inheritance mechanisms provided by KODIAK. This hierarchical organization of conventional metaphoric knowledge is the primary means used to capture the regularities exhibited by the system of metaphors in the language. Chapter 3 provides an analysis of some of these regularities and poses some representational requirements. Chapter 4 will demonstrate how these requirements are met in KODIAK.

1.3.2. Interpretation

Metaphor is a normal and conventional part of language. The interpretation of utterances containing metaphors should reflect this fact in the way that the metaphors are processed. In particular, the interpretation of metaphor should not be viewed as an exception to normal processing. The approach taken here is that metaphoric and literal interpretations have equal status, and are evaluated using interpretation mechanisms that are fundamentally the same.

The main thrust of this approach is that normal processing of metaphoric language proceeds through the direct application of specific knowledge about the metaphors in the language.

The interpretation of sentences containing metaphoric language is a two-step process in MIDAS. The first step in the interpretation of an input sentence is the production of a syntactic parse and a preliminary semantic representation. In the second step, this preliminary representation is replaced by the most specific interpretation that can coherently account for the input. This interpretation may be a literal one or one of a number of conventional metaphorical interpretations.

This general interpretation process has been implemented in the Metaphor Interpretation System (MIS) component of MIDAS. The MIS examines the initial primal representation in an attempt to detect and resolve uses of conventional UNIX metaphors. In the following UC example, a user has posed a question involving the conventional metaphor structuring processes as enclosures. The MIS component finds and resolves this metaphor. The resolution produces an instantiation of a coherent target concept representing the correct conventional meaning of the utterance.

In the following example, trace output is interspersed with a running commentary shown in normal Roman font. New examples are introduced with bold horizontal lines, while a running commentary is delimited with narrower lines.

```
> (do-sentence)
Interpreting sentence:

How can I get into lisp?

Interpreting primal input.

(A Entering50 (↑ Entering)
   (agent597 (↑ agent) (A I203 (↑ I)))
   (patient562 (↑ patient) (A Lisp58 (↑ Lisp))))
```

The input phrase *get into* is treated as a phrasal unit with a conventional meaning corresponding to `Entering`. The preliminary semantic representation produced in this step is called the *primal representation* (Wilensky 1987). The primal representation produced by the parser represents concepts derivable from knowledge of the grammar and lexicon available to the parser. In particular, the primary task accomplished in this phase is the appropriate assignment of filled case roles to the concept underlying the head of a phrase. This primal representation* represents a level of interpretation that is explicitly in need of further semantic processing. It should not be confused with what has traditionally been called a literal meaning. The primal representation should be simply considered as an intermediate stage in the interpretation process where only syntactic and lexical information has been brought to bear.

```
Concreting input relations.

  Concreting  patient to entered.
  Concreting  agent to enterer.
```

The `patient` and `agent` roles, with their respective filler concepts `I203` and `Lisp58`, were derived solely from the verb class that *enter* belongs to, and the syntax of the sentence. In this next step of processing, these generic roles are replaced by the more specific semantic roles that are actually attached to the `Entering` concept. An inference where a given concept is replaced by a more specific concept, based on partial information, is called a *concretion inference* (Norvig 1983, Wilensky 1983, Wilensky et al 1986). The details of concretion are given in Chapter 5. In this example, it is inferred

* This use of primal content differs from Wilensky's formulation in several ways. Wilensky (1987) does not envision these as separate processing stages at all, but rather as aspects of the analysis of an utterance. However, the process model developed in this thesis gives rise naturally to an intermediate stage of representation that does correspond to part of Wilensky's primal content. A more complete discussion of the role of the primal representation and its relation to Wilensky's is given in Chapter 5.

that the `agent` is an `enterer` and the `patient` of an `Entering` is an `entered`.

```
Interpreting concreted input.

(A Entering50 (↑ Entering)
   (enterer50 (↑ enterer) (A I203 (↑ I)))
   (entered50 (↑ entered) (A Lisp58 (↑ Lisp))))

Failed interpretation: Entering50 as Entering.
Failed interpretation: Entering50 as Enter-Association.
```

The literal interpretation and one of the other known Entering metaphors are rejected before the correct metaphor is found and applied. These interpretations are rejected because the input concepts filling the roles of `enterer` and `entered` do not match the requirements for these roles in these interpretations. In particular, the interpretation as an actual `Entering` requires that the `entered` concept must be a kind of enclosure. The filler of the `entered` role in the input, `Lisp58`, fails this requirement, therefore this interpretation is rejected. Similarly the `Enter-Association` metaphor specifies that the `entered` concept must be a kind of `Association`. Again, `Lisp58` fails to satisfy this constraint and causes the rejection of the metaphoric interpretation posing this constraint.

Note that the fact that the system considers the literal interpretation first is an artifact of the search procedure. It does not indicate any reliance on attempting the literal meaning first as was the case in previous approaches. All the conventional metaphorical uses have equal status with the known literal concept, `Entering`.

```
Valid known metaphorical interpretation.

Applying conventional metaphor Enter-Lisp.

(A Enter-Lisp (↑ Container-Metaphor Metaphor-Schema)
   (enter-lisp-res enter-res → lisp-invoke-result)
   (lisp-enterer enterer → lisp-invoker)
   (entered-lisp entered → lisp-invoked)
   (enter-lisp-map Entering → Invoke-Lisp))

Mapping input concept Entering50 to concept Invoke-Lisp30
  Mapping input role enterer50 with filler I203 to
       target role lisp-invoker30
  Mapping input role entered50 with filler Lisp58 to
       target role lisp-invoked30

Yielding interpretation:

(A Invoke-Lisp30 (↑ Invoke-Lisp)
```

```
   (lisp-invoked30 (↑ lisp-invoked)
    (A Lisp58 (↑ Lisp)))
   (lisp-invoker30 (↑ lisp-invoker)
    (A I203 (↑ I))))
```

The `Enter-Lisp` metaphor has been found and applied to the given input concepts. The main source concept is interpreted as an instance of the `Invoke-Lisp` concept according to the `enter-lisp-map`. The input roles `enterer` and `entered` are interpreted as the target concepts `lisp-invoker` and `lisp-invoked` respectively.

This interpretation of the `Entering` concept is then used to fill the role of the `topic` role of the How-Question that constitutes the representation of the rest of the sentence.

```
Final interpretation of input:

(A How-Q207 (↑ How-Q)
   (topic206 (↑ topic)
    (A Invoke-Lisp30 (↑ Invoke-Lisp)
       (lisp-invoked30 (↑ lisp-invoked)
     (A Lisp58 (↑ Lisp)))
       (lisp-invoker30 (↑ lisp-invoker)
     (A I203 (↑ I))))))
```

This how-question, with the reinterpreted topic concept, is then passed along to the next stage of UC processing. UC then prints the answer as follows.

```
Calling UC on input:

(A How-Q207 (↑ How-Q)
   (topic206 (↑ topic)
    (A Invoke-Lisp30 (↑ Invoke-Lisp)
       (lisp-invoked30 (↑ lisp-invoked)
     (A Lisp58 (↑ Lisp)))
       (lisp-invoker30 (↑ lisp-invoker)
     (A I203 (↑ I))))))

UC: You can get into lisp by typing lisp to the shell.
```

Note that when a conventional metaphor has been employed by the user in asking a question, UC's natural language generator uses the same metaphor in producing the answer. In this example, the system uses the same enclosure metaphor employed by the user to express the plan.

A complete description of the MIS component of MIDAS is given in Chapter 5. In particular, it describes how the MIS can perform the following tasks: find relevant candidate metaphors to apply, choose a correct one to apply based upon the constraints of the utterance, and instantiate a representation of the appropriate target concepts.

1.3.3. Learning

MIDAS will inevitably face the situation where a metaphor is encountered for which none of its known metaphors provides an adequate explanation. This situation may result from the existence of a gap in the system's knowledge-base of conventional metaphors, or from an encounter with a novel metaphor. In either case, the system must be prepared to handle the situation.

The approach taken here to the understanding of new or unknown metaphors is called the Metaphor Extension Approach. The basic thrust of this approach is that a new metaphor can best be understood by extending an existing metaphor in a systematic fashion. The basis for this approach is the belief that the known set of conventional metaphors constitutes the best source of information to use in understanding new metaphors.

The basic strategy is to first find a known metaphor that is systematically related to the new example. This candidate metaphor is then applied to the new example in an attempt to produce an appropriate target meaning. The process of applying the candidate metaphor to the new example is dependent upon the kind of semantic connection between the candidate and the new example. Three kinds of connections are recognized, yielding three kinds of extension inferences: *similarity extension*, *core-extension* and *combined-extension*. The details of these inferences are given in Chapters 7 through 9.

Once the intended target meaning of the new example has been determined, a new metaphor is created and stored away for future use. When metaphors of this type are encountered again the system can interpret them directly.

This strategy is realized in the Metaphor Extension System (MES) component of MIDAS. When no coherent explanation can be found for a given primal input it is passed along to the MES. The basic steps of MES algorithm are given in the following sections. These steps will then be made more concrete in terms of a detailed trace from the MES.

Step 1: Characterize the new input. Partial source and target components of a new metaphor are extracted from the primal representation accepted as input. The terms *current source* and *current target* will be used to refer to the source and target concepts derived from the input example.

Step 2: Search for related metaphors. This step searches for any known metaphors that are potentially related to this new use. The search consists of an attempt to find a path or paths through the network from the current source to the current target concepts that contains a known metaphor. A metaphor contained in such a path is judged to be relevant.

Step 3: Evaluate the set of candidate metaphors found in Step 2. The purpose of this step is to select a metaphor from the set found in Step 2 for further processing. This choice is based on a set of criteria to determine the metaphor that is closest conceptually to the current example.

Step 4: Apply this previously understood metaphor to the current example. The candidate metaphor is applied to the current target based on the relationship between the candidate mapping and the current example. Depending on this relationship, either a similarity, core, or combined extension inference is performed.

Step 5: Store the new metaphor. Create and store a new metaphor consisting of the source and target concepts identified in the above steps along with appropriate associations between them. This new metaphor will be used directly when future instances of this metaphor are encountered.

Consider the processing of the following example from the MES.

```
> (do-sentence)

Interpreting sentence:

How can I kill a process?

Interpreting primal input.

(A Killing16 (↑ Killing)
   (agent87 (↑ agent) (A I46 (↑ I)))
   (patient76 (↑ patient)
    (A Computer-Process10 (↑ Computer-Process))))
```

The parser accepts the input sentence, as specified by the user, and produces a primal representation of the input in the form of KODIAK concepts. This primal representation contains only information derivable from the grammar and lexicon. This primal representation is taken to be one that is in need of further elaboration and interpretation. In this example, the concepts associated with the verb *kill* are interpreted first.

```
Concreting input relations.

  Concreting  patient to kill-victim.
  Concreting  agent to killer.

Interpreting concreted input.
```

```
(A Killing16 (↑ Killing)
   (killer16 (↑ killer) (A I46 (↑ I)))
   (kill-victim16 (↑ kill-victim)
    (A Computer-Process10 (↑ Computer-Process))))
```

The case relations specified in the initial primal representation are replaced by more specific relations that are directly attached to the concept `Killing`. This is called the concreted primal representation. Note that the constraints on the more specific relations are not checked against the fillers of the roles during this concretion. In this example, this concretion results in the `agent` role being concreted to the `killer` and the `patient` role being concreted to the `kill-victim`.

```
Failed interpretation: Killing16 as Killing.
Failed interpretation: Killing16 as Kill-Delete-Line.
Failed interpretation: Killing16 as Kill-Sports-Defeat.
Failed interpretation: Killing16 as Kill-Conversation.

No valid interpretations. Attempting to extend existing metaphor.
```

Once the concreted representation has been created, it must undergo further processing to find a valid interpretation. The system attempts to determine if the given input is consistent with any of the known conventional interpretations, literal or metaphorical. In this case, the input is not consistent with either the literal `Killing` concept or any of the three known metaphorical uses of *kill*.

As in the previous example, the fact that the system considers the literal interpretation first is an artifact of the search procedure. The conventional metaphorical uses have equal status with the known literal concept, `killing`.

At this point, all the possible conventional interpretations of the primal input have been eliminated as potential readings. The input is now passed to the Metaphor Extension System in an attempt to extend an existing metaphor to cover this new use and determine the intended meaning.

```
============================================================
Entering Metaphor Extension System
============================================================

Searching for related known metaphors.

Metaphors found: Kill-Conversation Kill-Delete-Line Kill-Sports-Defeat
```

The first step in the extension step is to collect all the relevant known metaphors that might be related to this new use. This initial search scans through all the metaphors directly attached to the input concept, and also at all the metaphors attached to concepts that are core-related to the input concept. In this case, the system has knowledge of three metaphors that share the same source concept with the current use.

```
Selecting metaphor Kill-Conversation to extend from.

(A Kill-Conversation (↑ Kill-Metaphor Metaphor-Schema)
   (kill-c-res kill-result → conv-t-result)
   (killed-conv kill-victim → conv-termed)
   (killer-terminator killer → conv-termer)
   (kill-term Killing → Terminate-Conversation))
```

The candidate metaphors are ranked according to a "conceptual distance" metric. This is a measure of how close the candidate metaphors are to the new example. The primary factor contributing to this metric is a measure of similarity between the target concepts of the candidate metaphor and the input filler concepts. This conceptual distance metric is fully explained in Chapter 7. The candidate metaphor that is judged to be closest to the input example according to this metric is chosen as the candidate metaphor for further processing.

The selected metaphor is classified for further processing according to its relationship to the input example. In this case, the candidate metaphor is in a similarity relationship to the input metaphor.

```
Attempting a similarity extension inference.

Extending similar metaphor Kill-Conversation with
target concept Terminate-Conversation.

Abstracting Terminate-Conversation to ancestor concept
Terminating producing abstract target meaning:

(A Terminating3 (↑ Terminating)
   (terminated3 (↑ terminated)
    (A Computer-Process10 (↑ Computer-Process)))
   (terminator3 (↑ terminator) (A I46 (↑ I))))
```

The first step in the processing of a similarity extension inference is to identify the concepts specified in the input example with their corresponding target concepts. In this example, the concept `Computer-Process10` is identified with the target role of `terminated-conversation`, and the role of `I46` is identified with the target role of `converstation-terminator`. The constrainers of these concepts, however, are too specific to accept these input concepts. In this example, there is a mismatch between the input concept `Computer-Process` and the candidate target concept `Conversation`.

The next step, therefore, is to abstract the target concept of the candidate to the first concept that can accept the concepts specified in the input. In this case, the concept `Terminate-Conversation` is abstracted to its ancestor concept `Terminating`. The ancestor of the `terminated-conversation` role has as a constrainer the abstract concept `Process`, which can constrain the more specific concept `Computer-Process`.

```
Concreting target concept Terminating to Terminate-Computer-Process
producing concreted meaning:

(A Terminate-Computer-Process10
   (↑ Terminate-Computer-Process)
   (c-proc-termer10 (↑ c-proc-termer)
    (A I46 (↑ I)))
   (c-proc-termed10 (↑ c-proc-termed)
    (A Computer-Process10 (↑ Computer-Process))))
```

The next step in the similarity extension inference is to look down the hierarchy from `Terminating` to see if there are any more specific concepts beneath this one that can adequately accommodate the input concepts. The specific existing concept `Terminate-Computer-Process10` is found. The concept `c-proc-termed10` is a more specific concept than `terminated` and can still accept the input concept `Computer-Process10` as a filler, since the constraining concept on the concept `c-proc-termed` is a `Computer-Process`.

```
Creating new metaphor:

 Mapping main source concept Killing to main target concept
   Terminate-Computer-Process.
  Mapping source role killer to target role c-proc-termer.
  Mapping source role kill-victim to target role c-proc-termed.

(A Killing-Terminate-Computer-Process (↑ Kill-Metaphor)
   (kill-victim-c-proc-termed-map kill-victim → c-proc-termed)
   (killer-c-proc-termer-map killer → c-proc-termer)
   (killing-terminate-computer-process-map Killing
    → Terminate-Computer-Process))
```

The next stage of processing creates a new metaphor that represents this newly learned use. The role correspondences from the input example to the target concepts of the candidate metaphor form the basis for a new set of metaphoric associations that make up the new metaphor-sense. In this case, the main source concept, `Killing`, is mapped to the intended target concept `Terminate-Computer-Process`. The source aspectuals `killer` and `kill-victim` are mapped to the concepts `c-proc-termer` and `c-proc-termed`, respectively. In each case, a new metaphor-map is created to connect the source and target concept in the knowledge base. The map is then classified properly in the hierarchy of existing maps and connected to the newly created metaphor-sense representing this new metaphor.

In the case of a similarity extension inference, the newly created metaphor-maps are made siblings (children of the same parent) of the corresponding metaphor-maps from the candidate metaphor used. The newly created metaphor-sense is also made a sibling of candidate metaphor. In the current example, the newly created metaphor-sense, `Kill-Terminate-Computer-Process`, is made a sibling of the candidate metaphor `Kill-Conversation`. The names given to the new metaphor-maps and metaphor-sense are created by concatenating the names of the component source and target concepts. (Although the names themselves are not relevant to the functioning of the program)

```
Final interpretation of input:

(A How-Q46 (↑ How-Q)
   (topic46 (↑ topic)
    (A Terminate-Computer-Process10
       (↑ Terminate-Computer-Process)
       (c-proc-termer10 (↑ c-proc-termer)
     (A I46 (↑ I)))
       (c-proc-termed10 (↑ c-proc-termed)
     (A Computer-Process10
        (↑ Computer-Process))))))
```

The final representation of the input sentence now contains the intended target concept, `Terminate-Computer-Process`, as the topic of user's original how-question.

```
Calling UC on input:

(A How-Q46 (↑ How-Q)
   (topic46 (↑ topic)
    (A Terminate-Computer-Process10
       (↑ Terminate-Computer-Process)
       (c-proc-termer10 (↑ c-proc-termer)
     (A I46 (↑ I)))
       (c-proc-termed10 (↑ c-proc-termed)
     (A Computer-Process10
        (↑ Computer-Process))))))

UC: You can  kill a computer process by typing ^c to the shell.
```

In the case where the system is processing a UC question, the final representation of the input concepts are then passed to the UC system for further processing. In this case, the system proceeds to answer the user's question on how to terminate a process. In the event that the main concept that UC is addressing has been derived metaphorically, UC's generator uses the user's source language when expressing its answer. In this case, the generator adopts the user's use of *kill* to describe the termination.

The following session demonstrates the altered processing by the system now that the `Killing-Terminate-Computer-Process` has been acquired. The same question is again posed to the system.

```
> (do-sentence)

Interpreting sentence:

How can I kill a process?

Interpreting primal input.

(A Killing17 (↑ Killing)
   (agent88 (↑ agent) (A I47 (↑ I)))
   (patient77 (↑ patient)
    (A Computer-Process11 (↑ Computer-Process))))

Concreting input relations.

  Concreting  patient to kill-victim.
  Concreting  agent to killer.

Interpreting concreted input.

(A Killing17 (↑ Killing)
   (killer17 (↑ killer) (A I47 (↑ I)))
   (kill-victim17 (↑ kill-victim)
    (A Computer-Process11 (↑ Computer-Process))))


Failed interpretation: Killing17 as Killing.

Valid known metaphorical interpretation.

Applying conventional metaphor Killing-Terminate-Computer-Process.

(A Killing-Terminate-Computer-Process (↑ Kill-Metaphor)
   (kill-victim-c-proc-termed-map kill-victim → c-proc-termed)
   (killer-c-proc-termer-map killer → c-proc-termer)
   (killing-terminate-computer-process-map Killing
    → Terminate-Computer-Process))
```

The application of this known metaphor immediately yields the intended interpretation.

```
Yielding interpretation:

(A Terminate-Computer-Process11
   (↑ Terminate-Computer-Process)
   (c-proc-termed11 (↑ c-proc-termed)
    (A Computer-Process11 (↑ Computer-Process)))
   (c-proc-termer11 (↑ c-proc-termer)
    (A I47 (↑ I))))

Concretion yields:

(A Terminate-Computer-Process11
   (↑ Terminate-Computer-Process)
   (c-proc-termed11 (↑ c-proc-termed)
    (A Computer-Process11 (↑ Computer-Process)))
   (c-proc-termer11 (↑ c-proc-termer)
    (A I47 (↑ I))))
```

As in the previous example, all the conventional meanings are attempted before an interpretation is settled upon.

```
Failed interpretation: Killing17 as Kill-Delete-Line.
Failed interpretation: Killing17 as Kill-Sports-Defeat.
Failed interpretation: Killing17 as Kill-Conversation.

Final interpretation:

(A Terminate-Computer-Process11
   (↑ Terminate-Computer-Process)
   (c-proc-termed11 (↑ c-proc-termed)
    (A Computer-Process11 (↑ Computer-Process)))
   (c-proc-termer11 (↑ c-proc-termer)
    (A I47 (↑ I))))

Calling UC on input:

(A How-Q47 (↑ How-Q)
   (topic47 (↑ topic)
    (A Terminate-Computer-Process11
       (↑ Terminate-Computer-Process)
       (c-proc-termed11 (↑ c-proc-termed)
     (A Computer-Process11
        (↑ Computer-Process)))
```

```
    (c-proc-termer11 (↑ c-proc-termer)
  (A I47 (↑ I))))))
```

```
UC: You can kill a computer process by typing ^c to the shell.
```

1.4. Summary

The approach taken to metaphor in this thesis is fundamentally a knowledge-based one. The metaphors that are a conventional part of the language are represented directly as associations between source and target concepts. The normal interpretation of metaphoric language proceeds through the application of this metaphoric knowledge. Finally it is shown how new metaphors can be learned by incrementally extending these known metaphors.

The rest of this thesis provides motivations for and descriptions of the various components of MIDAS. Chapter 2 presents an analysis of the relevant previous research that led to MIDAS. An analysis of the structure of both individual conventional metaphors and systems of metaphors is given in Chapter 3. This serves as a motivation for the representation described in Chapter 4. Chapter 4 serves to introduce the general features of the KODIAK representation language along with the specific details of KODIAK that are used to represent metaphoric language. A description of the metaphor interpretation component of MIDAS is given in Chapter 5.

The remainder of the thesis is devoted to the learning component of MIDAS. Chapter 6 gives an overview of the Metaphor Extension System (MES), which can acquire new metaphors when new unknown metaphors are encountered during normal processing. The complete details of the MES algorithm are given in Chapter 7. Chapters 8 and 9 provide the details for the two basic extension inferences used by the MES. Chapter 10 provides a set of extended examples illustrating how the MES can learn metaphors that were handled by previous computational approaches. Finally, some conclusions are given in Chapter 11.

Chapter 2

Related Research

This chapter reviews some of the relevant past research on computational approaches to metaphor. Much of the work discussed here is based upon the modern view of metaphor articulated by Richards (1936) and Black (1954, 1962). The collections of Ortony (1979) and Johnson (1981) provide an appropriate review of the relevant metaphor literature.

As indicated in Chapter 1, the primary difference between the approach taken in this thesis and most of the previous computational approaches has to do with the use of explicit metaphoric knowledge. This chapter will focus on the relationship between the strategies employed in these earlier systems and the use of metaphoric knowledge. One of the main results will be to show that the most successful systems derived their power from strategies that made implicit use of knowledge of conventional metaphor.

2.1. Knowledge-Deficient Approaches

The principle characteristic of the approaches described in this section is that metaphor is treated as an anomalous novel departure from conventional language. Because metaphor is not considered to be a conventional part of the language, explicit knowledge about the metaphors in the language is not available to these approaches. This complete lack of explicit knowledge about the metaphors in the language leads to the term knowledge-deficient.

These approaches do, however, make use of arbitrary amounts of knowledge about context and the semantics of the various source and target domains of the metaphors that they are concerned with. This knowledge is used in a variety of ways to determine the meaning of metaphors when they are encountered.

2.1.1. Natural Language Processing Approaches

The approaches described in this section are explicitly concerned with natural language processing concerns. That is, they are actually concerned with the task of interpreting sentences containing metaphors.

The following approaches all follow what I call the "literal meaning first" approach. This approach asserts that the only conventional meaning is the literal meaning. These systems all attempt to first interpret their inputs in terms of this literal meaning. A metaphorical interpretation is only attempted if the input is clearly not compatible with a literal interpretation. This incompatibility is typically manifested by a violation of one or more of the semantic constraints posed by the literal interpretation, usually in the form of selection restriction violations.

The presence of a selection restriction violation is taken as evidence for the existence of metaphor. At this point, these approaches take a variety of approaches to dealing with this situation depending on the particular concerns of the individual research.

2.1.1.1. Wilks

Wilks (1973, 1975, 1978) presents a series of approaches to metaphor centered around the notion of *preference semantics*. This work developed partially as a reaction to the work on lexical semantics by Katz and Fodor (1963) which gave rise to the idea of selection restrictions. According to Katz and Postal selection restrictions could be used on a binary basis to determine the well-formedness of particular utterances. Consider the following example from Wilks (1975):

(1) My car drinks gasoline.

According to the selection restriction approach, the verb *drink* requires that the drinker be an animate. The above sentence is simply not well-formed because the actor is a car and not animate. Wilks points out that such selection restriction violations are the case more often than not, and that the utterances in which they occur are perfectly interpretable. He therefore proposed the notion of a *preference*. As far as metaphor was concerned, the initial use of preferences was to simply recognize a potential metaphorical use when a preference restriction was violated. However, no attempt was made to find the intended target meaning of the utterance. This technique of using selection restriction violations to detect metaphors was adopted in almost all subsequent computational approaches.

In Wilks (1978) the notion of simply accepting a metaphorical preference violation was replaced with the notion of a *projection* using a knowledge structure called a pseudo-text. A pseudo-text was a script-like representation of some episodic or contextual information about the target domain. The process of projection replaced the item with the violated preference with some other concept from the pseudo-text.

Consider the operation of a projection inference on Example (1). The *drink* predicate would be replaced by a concept from a pseudo-text containing detailed information about the target concept car. The replacement concept is chosen by matching the source concept *drink* against all the concepts in the car pseudo-text. The concept that best matches is chosen as the replacement. In this case, the concept of a car *using* gasoline is chosen as the target concept that best matches *drink*.

It is hard to evaluate this approach since the details of the general matching algorithm are never given. Another problem is that the semantic status of the pseudo-texts is never suitably explained. In particular, it is not clear what goes into a pseudo-text and how it relates to more general long-term semantic information. Consider the following problem posed by Wilks.

(2) Britain entered the Common Market.
(3) Britain tried to leave the Common Market.

Example (2) can be handled perfectly well by the projection mechanism. A pseudo-text about the Common Market is accessed and the source concept, ***entering***, is matched against it yielding a concept representing the idea of Britain associating itself with the Common Market. However, Wilks claims that (3) could not be handled because no pseudo-text could exist to represent the idea of Britain leaving the Common Market, since this event has never occurred. This seems to imply that pseudo-texts are episodic in nature and not semantic. Surely a system that can understand the notion of joining an association, should be capable of understanding the notion of quitting. In any case, the general approach of replacing the source concept with a concept related to the input target concept is basically sound. The problem with the projection inference is the lack of detail in the matching algorithm and the vagueness in describing from where candidate target concepts are chosen. These two issues will reappear in the discussions of the rest of the previous approaches.

Another problem with this approach, and with all the selection restriction approaches, is the reliance on selection restriction or preference violations. As Wilks states "projection is operated only in the presence of preference-breaking". Consider the following example.

(4) McEnroe killed Connors.

This example involves no violations of the literal semantics of *kill* and yet has an obvious conventional metaphorical meaning. Similarly a literal interpretation of *no man is an island* involves no semantic violations. A literal interpretation of either of these examples does involve what might be called appropriateness conditions. In these examples, these conditions might be that it would be unlikely that McEnroe would actually slay Connors, or that literally stating that *no man is an island* adds no new information. In general, the literal interpretation of a metaphorical utterance will cause a violation of one or more of these appropriateness conditions. These may or may not include a violation of a selection restriction. This fact poses serious problems for theories of metaphor that rely heavily on the belief that metaphors arise exclusively from selection restriction violations of the literal interpretation. Some possible solutions to this problem will be discussed in Chapter 5.

This strategy is in one form or another the fundamental one used in all the following approaches. It can be summarized in the following two steps.

1 **Detect** the presence of a metaphor by noticing a selection restriction violation.

2 **Replace** the source concept with some known concept that is related in some way to the concept causing the violation.

The focus in the following discussions will be on the second step. Of particular interest will be the workings of various matching algorithms that replace the source concept with an intended target by directly matching the source against all possible target concepts.

2.1.1.2. Fass

Fass (1983, 1987, 1988) introduces an approach the lexical semantics called Collative Semantics (CS). CS builds on and extends the preference semantics approach of Wilks. Fass addresses a wide range of lexical issues including metaphor. Consider Fass's reanalysis of the following example from Wilks.

(6) My car drinks gasoline.

Fass, like Wilks, adopts the basic approach of noticing a metaphor by virtue of a selection restriction violation and then replacing it with some target concept. As mentioned above Wilks provided no principled method for matching source concepts against various targets in order to select the appropriate meaning. Fass provides the details of an approach to this problem by using an abstraction hierarchy. The basic assertion is that the intended target meaning will be a sibling of the source concept in an abstraction hierarchy. The search for this sibling is guided by the target concepts in the input example.

Consider the details of (6). The definition of the source concept *drink* requires that an animate drinker consume a potable liquid. The input sentence violates these restrictions because a car is not an animate and gasoline is not a potable liquid. These violations suggest a metaphorical reading.

The intended target meaning of the sentence is found by matching the facts of the long-term source concepts (that an animal drinks liquids) against all the known facts about the target concept car. This analogical matching proceeds by attempting to find common ancestors between the source concept and the various concepts in the target domain. In this case, we find that *drink* in the source matches *use* (cars use gasoline) in the target domain. This match is successful because there is a common ancestor concept, *expend*, between *drink* and *use*. This analogical match is confirmed because a common ancestor match between *gasoline* and *potable-liquid* is coherent with the semantics of *expend.* The meaning of the metaphor is, therefore, represented as an analogy of the form that animals drink potable liquids as cars use gasoline.

The approach just described works well when the source and target concepts of the metaphor are closely related hierarchically. The greater the conceptual distance between the source and target concepts, the smaller the likelyhood that the common ancestor concept will provide a meaningful analogy. Nevertheless, the use of abstraction hierarchies to perform matching operations at least provides a basis for the matching of distinct source and target concepts. Chapter 8 discusses the limitations of using abstraction hierarchies to do this kind of analogical learning. In addition, it will show that the usefulness of this matching approach is greatly amplified by the addition of explicit conventional metaphorical knowledge.

2.1.1.3. Russell.

The system described in Russell (1974) fits in neatly with the approach just described. Russell uses Conceptual Dependency (Schank and Abelson 1977) as her underlying knowledge representation. Violations of selection restrictions were modeled as violations of the restrictions on the various roles in CD primitives. Consider the following Russell example:

(5) She offered him an idea.

According to Russell, the essential source concept in this example involves a hypothetical PTRANS. (An offer of transfer of physical possession). However, the concept, *idea*, filling the role of the object transferred, violates the requirements for that role imposed by the PTRANS primitive. The filler of the transferred role must be a physical object. The intended target concept involves an MTRANS, representing the communication, at some level, of the idea from the actor to the recipient.

Russell demonstrates how a structure matching system can exploit the structure of these violations to actually determine the intended target meaning of the metaphor. Given the simplicity of the described matching algorithm, the system demonstrated surprisingly successful results over a certain class of metaphors. An analysis makes it clear that the system made widespread implicit use of conventional metaphoric knowledge to achieve these results.

Russell employed two techniques that in reality comprise implicit knowledge of conventional metaphor. The first technique used a table of selection restriction violation types. This table associated certain kinds of restriction violations with certain target domains. This table was used to limit the number of concepts that had to be matched during the matching phase. For example, the table indicated that when a constraint requiring a physical object was violated by some kind mental object, as in (5), then the target concept was to found among a set of concepts marked as being in the mental domain. The original source concept was then only matched against concepts in this domain. This represented an implicit set of conventional associations from a source to a target domain.

The second source of implicit metaphoric knowledge is much more important to the success of this program. The target concepts of the various metaphors discussed were implicitly structured in terms of the original source concepts. The reason that the matcher could conclude a correct match among many possible candidates, using only a simple matching algorithm, results from this implicit structuring of the target domains.

Consider the MTRANS CD primitive which matches the PTRANS source concept in (5). Russell's matcher finds that MTRANS has a high degree of match with PTRANS. This is apparently based on the similar case structure of these primitives. In particular, each primitive possesses an *ACTOR*, *OBJECT*, *FROM* and *TO* case role that are allowed to match as identical roles. This structure, however, obscures the fact the MTRANS primitive is nothing more than a primitive codification of the Conduit Metaphor, (Reddy 1979). This metaphor entails, in part, that ideas are objects, possession is belief or knowledge, and that transfer of these ideas is communication. The MTRANS primitive simply embodies this metaphor.

The power of this approach therefore arises not from its matching capabilities. Rather, the power arises from the fact that the target concepts have been structured in terms of a deeply embedded set of conventional conceptual metaphors. Viewed in these terms, the overall algorithm of searching for the best match seems a little backwards. Rather than hiding the conventional metaphor and making the system search for it with an ad-hoc search, it would seem to make more sense to make the metaphor explicit and allow it to be indexed and recognized directly.

2.1.2. Analogical Approaches

The approaches described in this section are not primarily interested in natural language processing. Rather, most of these research efforts are concerned with analogical reasoning processes. Most make the assertion that while they are not explicitly dealing with metaphor their approaches nevertheless extend to the interpretation of metaphor. The implicit assertion is that the solution to the problems of interpreting and producing metaphors will fall out from their more general solutions to analogical reasoning.

In particular, these approaches assert that the meaning of a metaphor results from an analogically directed transfer of relations from the source to the target domain. This is different from the approach illustrated above by Fass. In that approach an analogical match was used to select among existing candidate target concepts. The approaches described in this section are actually creating new meanings by transferring relations into the target domain from the source domain.

The analogical transfer approach must perform four tasks in order to succeed in properly interpreting a metaphor. These task are as follows:

Task 1: Circumscribe the source and target domains. The concepts that actually constitute the source and target domains must be identified.

Task 2: Establish object correspondences. A coherent set of concept to concept correspondences must be established from the source to target domain.

Task 3: Select the transfer set. A set of relations in the source domain must be chosen for transfer to the target domain.

Task 4: Perform the transfer. The final step is transferring the selected relations in such a way that they are consistent with the semantics of the target domain.

There are difficult unsolved problems with each of these tasks. None of the following approaches have successfully found a solution to all of these tasks. One or more of the tasks is either finessed or ignored completely. This fact limits the practical application of any of these approaches to the actual interpretation of metaphorical natural language inputs. However, these approaches do provide a number of insights that can be usefully applied. In particular, Chapters 8 and 9 will show that by properly using existing metaphorical structures and some of the heuristics discovered in these approaches the complexity of the analogical approach can be reduced to the point where it is a practical.

2.1.2.1. Winston and Carbonell

Winston (1980) and Carbonell (1981) make a similar set of proposals that basically address Task 3. The basis of these approaches is an analysis of various analogies and metaphors. This analysis yields the insight that certain kinds of predicates tend to be transferred from the source to target domains in many different analogies and metaphors. This leads to the strategy that Task 3 can be solved by simply searching for an a priori list of relations and predicates in the source, and then transferring them directly to the target. Winston focusses on the preservation of *CAUSAL* relations while Carbonell introduces an ordered list of preserved relations.

Both of these approaches solve Task 4 by transferring relations from the source to the target identically. It turns out that this strategy works well in these approaches. This is because the proposed set of preserved relations in both of these approaches are abstract enough (like *CAUSE)* to be applied across a wide range of domains. The approaches do not attempt the transfer of systematic domain specific relations.

2.1.2.2. Gentner

Gentner's (1983) structure mapping theory proposes a solution to the general problem of analogical transfer. In fact, only Task 3 is reasonably addressed.

Task 1 is solved by simply assuming that well-specified source and target concepts are provided as input. The problem of finding an appropriate analog in not addressed. More importantly, the problem of circumscribing the target domain is also not addressed.

Task 2 is not addressed in Gentner's original work but is addressed in a later computer implementation in Falkenhainer (1986). An object correspondence is found by computing all possible pair-wise combinations of concepts and choosing the best one based on a ranking of each arrangement. Indurkyha (1987) makes a similar proposal and points out that his approach his computationally intractable. The obvious problem with these approaches is the exponential number of possible configurations that have to be considered. If there are N concepts in the source and target domains then there are N! possible configurations to consider. This approach is obviously intractable in any domain with a reasonable number of concepts. Any analogy approach must, therefore, be augmented with additional domain specific knowledge that limits the number of configurations considered to control this explosive growth.

The main contribution of the structure mapping approach comes from its approach to the third task. Gentner proposes the *Systematicity Principle*, which states that in analogical transfers systematic sets of relations are preferred for transfer. In effective analogies, simple attributes of source concepts are rarely transferred, while relations among concepts are more likely to be preserved. Most likely to carry over are systematic sets of relations. A systematic set of relations in this theory refers to sets of relations that make reference to one another. This is a purely syntactic notion. Unlike the theories proposed by Winston and Carbonell, this theory makes no reference to particular domain specific relations that are frequently preserved. The intent is to provide a syntactic rule that will capture the same set of things that are captured by Winston's and Carbonell's list. This is accomplished by getting at the underlying structure of the relations that they merely list.

In Gentner (1983b, 1986), psychological evidence is given that people's judgements of the goodness of metaphors and analogies matches well with the constraints posed by the Systematicity Principle. In particular, people find most effective those analogies and metaphors that carry over systematic relational information.

Unfortunately, the Systematicity Principle is not by itself useful in understanding metaphors. This is because it merely states that relations that are carried over will be systematic. Note, however, that it does not state that all systematic relations should carry over. In the case of metaphor, it is usually the case that only a small subset of the source domain is relevant to the structure of the target domain. This subset will be systematic but there may be many systematic pieces of the source domain that are not relevant to the metaphor and are not carried over. Therefore, while the Systematicity Principle imposes a useful constraint on the transfer of relations, it is not by itself sufficient to determine an appropriate target meaning.

2.1.2.3. Greiner and Burstein

Greiner (1985) and Burstein (1986) arrive at similar solutions in their attempts to solve some of the complexity problems inherent in both the general best-match approach, and in Gentner's formulation. The general character of their solutions asserts that larger pieces of knowledge are needed. In particular, both use large pieces of knowledge that loosely correspond to the idea of chunks, scripts, or macro-operators. The intuition is that an understanding of the source or base domain consists not only of a logical set of systematically inter-related predicates, but rather that there a larger chunks of knowledge available that represent sets of related predicates. These larger chunks of knowledge are used in a top-down manner to constrain the number of concepts considered in determining object correspondences, and also in determining what predicates are transferred to the target domain. It is, therefore, the existence of these larger structures in the source analog that makes the task manageable.

Greiner presents the problem of trying to calculate the flowrate of a liquid through a pipe by using the analogical hint that "Flowrate is like current in an electric circuit". The problem is to determine which of the many facts associated with currents in an electric circuit are applicable to this problem. In particular, we would like to determine that the Ohm's law and Kirchoff's First and Second laws are relevant, while all the other facts known about electricity are irrelevant. Greiner postulates the prior existence of what he terms an *abstraction*. This abstraction is a set that includes only those concepts relevant to this example. The problem is, therefore, to use the analogical hint to find a relevant abstraction. This pre-existing abstraction serves to form the object correspondence, and to circumscribe the transfer set. Greiner proposes that useful abstractions are formed naturally as a by-product of previous problem-solving behavior.

Greiner's abstraction-based approach is fundamentally compatible with certain aspects of the MIDAS approach. In particular, the core-structuring of conventional metaphors, which arises from the sharing of important core concepts, corresponds to Greiner's notion of an abstraction. Chapters 3 and 9 show how these core-structures can be used to form an initial object correspondence and to guide the transfer of relations from the source to target domain. In addition, it will be shown that this sharing of important core concepts provides the motivation for the existence of structures like Greiner's.

The basic result of the analysis of these analogy approaches is that analogy can be a reasonable tool to use in the interpretation of metaphor. However, it is useful only if it is tempered with domain specific knowledge to make the approach tractable. In particular, Chapters 8 and 9 will show how explicit knowledge of existing conventional metaphors provides the leverage necessary to make analogy a reasonable approach to the interpretation of new metaphors.

2.2. Knowledge-Based Approaches

The metaphoric knowledge approach taken in this thesis was inspired by the work of Lakoff and Johnson (1980) and builds on the computational work of Jacobs (1985), Wilensky (1986) and Norvig (1987). Lakoff and Johnson effectively demonstrated the pervasive systematic conceptual nature of metaphor in language. They argue persuasively that metaphors are not a matter of novelty, but are rather a conventional part of our conceptual system.

The knowledge-based computational approaches described here have been influenced by recent linguistic approaches to lexical semantics. The work of Lindner (1981), Brugman (1981) and Herskovits (1986) are studies of the semantic interrelationships among various polysemous senses of English prepositions. These studies have drawn heavily from the works of Talmy (1975), Fillmore (1982), Lakoff (1987), and Langacker (1987). Rather than merely asserting that there are separate homonymous senses of these words, these researchers have attempted to formulate a coherent semantic structure that accounts for the many related uses. One of the primary findings has been that conventional metaphor is one of the major phenomena underlying the structure of the lexicon.

The work of Jacobs (1985) was the first computational approach that used the systematic representation of metaphoric knowledge to accomplish a natural language processing task. Jacobs used the explicit representation of metaphoric knowledge to generate sentences containing conventional metaphors. Consider the following examples.

(7) Ali gave Frazier a punch.
(8) Ali took a punch from Frazier.
(9) John gave Mary a kiss.
(10) John gave Mary a massage.

Each of these examples uses the idea of a transfer (giving and taking) to express concepts that actually involve no real transfer. Jacobs was concerned with developing a representation that could efficiently capture the generalizations exhibited by these kinds of examples, and also be used to generate examples like these when the target concepts themselves do not involve a transfer. Jacobs proposed that an explicit knowledge structure be used to represent the following conventional metaphorical facts.

Actions may be viewed as transfer-events with the actor playing the role of source, the object playing the role of the recipient, and the action itself playing the role of object.

Norvig (1987) used similar metaphorical structures in his FAUSTUS text-inferencing system to demonstrate how metaphorical utterances could be successfully interpreted using metaphorical associations. However, his system only applied this metaphorical knowledge when the KODIAK equivalent of a selection restriction violation was detected in a literal interpretation. Therefore, even though his system had knowledge of conventional metaphors, it was still subject to some of the same problems of previous systems in detecting metaphors in the absence of constraint violations.

The representation used in both of these efforts centered around the use of a set of hierarchically organized structured associations between the various source and target concepts of the conventional metaphors in the language. These structured associations were implemented using the VIEW mechanism of KODIAK (Wilensky 1986). A VIEW is a means of representing non truth-conditional facts as an explicit part of the knowledge that is represented about concepts in the knowledge base. These conventional conceptual metaphors are, therefore, directly represented as part of what is known. The representation used in this thesis is an extension to, and refinement of, these structured associations. Chapter 4 will give the details of the structures used in this thesis, and how they differ from these previous formulations.

2.3. Word Sense Acquisition

There have been a number of computational research efforts concerned with the problem of learning the meaning of new words. While none of these efforts directly address of the problem of metaphor, the approach they have taken is related to some metaphor understanding efforts. The efforts of Granger (1977), Selfridge (1981), and Zernik (1987) have all been attempts to learn the meaning of new words and phrases by using the context in which the use occurred. Each of these approaches had a different focus. Granger was concerned with determining the meaning of an unknown word during the script-based processing of a story. Selfridge was concerned with modeling early stages of language acquisition in children. Zernik's system acquired a hierarchical phrasal lexicon capturing idioms.

What these efforts share is the way in which context is used. In each case, the learning program was provided with an unknown word or phrase and a conceptual representation of some piece of text which already *includes the meaning of the unknown word*. The fundamental assumption of these efforts is that the learner has been able to determine the meaning of the word by some extra-linguistic means. The problem for the learner is to produce a pairing that links the unknown word to the correct concept already active in the context. This approach is similar to that used by Wilks (1978) for determining the meaning of metaphorical uses of known words using his pseudo-texts.

The approach taken in this thesis to learning new metaphors deliberately avoids the use of this kind of unrestricted contextual information. MIDAS relies on the semantic structure of its existing metaphoric and conceptual knowledge to determine the possible meaning of a new use. Chapter 6 gives the full rationale behind this approach and how it relates to these context approaches.

Chapter 3

Conventional Metaphoric Knowledge

3.1. Introduction

A major claim of this thesis is that knowledge about conventional metaphors should be represented declaratively. This chapter is an analysis of conventional metaphors that will illustrate exactly what needs to be captured by such a representation. Chapter 4 will describe the actual knowledge representation used to capture these facts.

This chapter will first discuss the basic requirements for representing individual conventional metaphors. It will then go on to show that our knowledge of the metaphors in the language cannot be adequately captured by merely listing the details of each one individually. The set of metaphors in the language, taken as a whole, demonstrates certain regularities. In order to see these regularities, groups of related examples must be analyzed. This chapter will, therefore, consist of a series of analyses demonstrating these regularities.

The format of this chapter consists of a series of analyses of the regularities exhibited by various conventional metaphors. For each phenomenon, a set of examples is given; a statement and analysis of the phenomenon in terms of its implications for a representation is given, and finally some suggestions are made for a representation that might adequately account for this regularity.

3.2. Conventional Metaphorical Meaning

A conventional metaphor consists of source and target concepts. The target concept is the concept that is actually under consideration. The source specifies the concepts in terms of which the target is being viewed. Consider the following example.

(1) Mary gave John a cold.

This example involves the conventional metaphor that to give someone a cold means to infect them with a cold. This metaphor will be referred to as the Infecting-As-Giving metaphor. The target concepts of this metaphor involve colds and the actions and events that lead to them. The source concept is giving. What is required is a way of determining the proper correspondence between these source and target concepts. In this case, there must be a way of representing the fact that the giving refers to an infecting, and not to any of the other many concepts associated with colds. This correspondence can be accomplished by directly representing an association between these concepts in the knowledge base.

This single association, between the main source and target concepts, is not sufficient by itself to capture all that needs to be captured. The concepts of giving and infecting are complex concepts consisting of many sub-parts. For example, a giving may be said to be an action involving a giver, a recipient, an object given, and some results. The primary results are that the recipient now has the given object and that the giver no longer has possession of the object. Similarly, the infecting concept has an infected person, the person who was the source of the infection, the infection itself, and the state of being infected. A set of associations are needed to provide the proper correspondences between the various sub-parts of the designated source and target concepts. In this case, the main association is from the concept giving to the concept of infecting. Associations related to this main one are: the giver is the infector, the recipient of the giving is the person infected, the object transferred plays the role of the infection itself and the result of the giving (the recipient possessing the object) is the result of the infecting.

The fundamental requirement of the representation, therefore, is to be able to represent these sets of explicit associations between complex source and target concepts. The representation must capture both the individual associations and the grouping of these associations into coherent metaphors. This requirement is summarized as follows.

> **Fundamental Representational Requirement:** Conventional metaphors must be explicitly represented as coherent sets of associations between source and target concepts.

The following sections will examine some of the regularities that are exhibited among groups of metaphors. The ability of the representation to capture these inter-metaphor regularities will flow from the representation of the conceptual associations used to satisfy the Fundamental Representational Requirement.

3.3. Extended Metaphors

Consider the following example:

(2) Mary has a cold.

Among the source target associations underlying this metaphor are the associations that to be infected with a disease is to possess it and that the infection is an object. These same associations were proposed as a part of the explanation for (1). The perceived relationship between these two metaphors arises from the fact that the definition of the Infecting-As-Giving metaphor includes the Infected-State-As-Possession metaphor as an important component part.

It seems, in fact, that the Infected-State-As-Possession metaphor is in a sense the more fundamental of these two. It is simpler and can stand on its own without the Infecting-As-Giving metaphor. It can for example be used to express infection with diseases for which there is no notion of transfer, as in (3).

(3) Mary has Alzheimer's disease.

However, the Infecting-As-Giving metaphor is completely dependent on the Infected-State-As-Possession metaphor in order to make sense. It seems, therefore, that the Infected-State-As-Infection metaphor motivates the corresponding transfer metaphor underlying (1).

The following sections will show that this kind of inter-metaphor relationship arises from the structured sharing of sub-parts by the related metaphors. In particular, it will show that sets of related uses like (1) and (2) always seem to share a more simple common metaphor.

3.3.1. Core Structures

Consider the following examples.

(4a) John *gave* Mary a book.
(4b) Mary *has* a book.
(4c) Mary *got* a book from John.

(5a) Mary *entered* the room.
(5b) Mary is *in* the room.
(5c) Mary *exited* the room.

(6a) Mary *killed* John.
(6b) John *died.*

(7a) Mike *opened* the door.
(7b) The door is *open.*
(7c) Mike *closed* the door.

The italicized words within each example make reference to a closely interconnected set of underlying concepts. Words like these, whose meanings are closely related, are said to be *core-related.* Intuitively, words are said to be core-related if their meanings make reference to a common set of concepts. This section will establish the requirements for determining core-relationships among concepts.

Consider the concepts underlying the use of the words *give, have,* and *get* in Example (4). The meaning of the word *have* as used in (4b) refers to a state of being in possession of some object. The meaning of *give* in (4a) refers to an action that results in a possession of an object. Similarly the use of *get* in (4c) also results in a possession but focuses more on the event as something happening to the recipient rather than an action by the original possessor. (The use of *get* in (4c) is actually ambiguous. The reading referred to here can be paraphrased as *received from*, as opposed to the sense meaning *went and got from*). Clearly, the meanings of both *give* and *get* make direct reference to the notion of possession underlying the use of *have* in (4b).

In a sense, the concept of possession underlying *having* is wholly contained within the concepts *giving* and *getting*. This reflects the underlying fact that an adequate understanding of the concepts of giving and getting presumes an understanding of the concept of possession. This kind of direct reference to, and joint sharing of, core concepts like having will be used to account for the perceived closeness of the words in all of the above examples. In particular, it lies at the core of the definition of what it means to be core-related.

Formally, two words can be said to be core-related if the following two conditions hold:

Relation Condition: The definition of one of the words makes direct reference to the definition of the other word.

Containing Condition: All of the sub-parts of the referenced definition are contained as sub-parts of the referring definition.

We can now say that *give* and *have* are core-related because both of these conditions hold. The Relation Condition holds because the concepts underlying *give* specify that the result of a giving is a having. The Containing Condition holds because all the sub-parts of having play integral roles in the definition of giving. In particular, the recipient of a giving is a haver and the given object in a giving is a thing had. A similar analysis accounts for the core-relatedness of *get* and *have*. In examples (4) through (7) the following pairs of words satisfy these two requirements: *give* and *have*, *get* and *have*, *enter* and *in*, *exit* and *in*, *kill* and *die*, *open* and *opened*, and finally *closed* and *open.*

Note that the analyses of the core-relations among *enter, exit, in*, and *out* is slightly more complex than the possession examples. The notions of being enclosed or unenclosed (in or out) relative to some enclosure are equally fundamental. The concepts underlying the words *enter* and *exit* satisfy the Containing Condition with both the idea of being enclosed and unenclosed. This example illustrates the case where a set of closely related core concepts underlies a set of core-related concepts. A similar situation holds for the *open* and *close* examples relative to the equally core concepts of being in an open or closed state.

Finally, consider the relationship between the meanings of the words *give* and *get*. The concepts underlying *give* and *get* are not directly related in the same way that *give* and *have* are. Instead, they are indirectly related by virtue of the fact that they have the concept having in common. They are both directly related to having, and contain it as a part of their individual definitions. This sharing of important sub-concepts yields a relationship called a *intermediate core-relation*. Two words may be said to be intermediate core-related if they are both core-related to the same third concept. In this case, *give* and *get* are both core-related to *have* since they satisfy both the Relation and Containing conditions. This sharing yields the following condition that must be satisfied for intermediate core-relatedness.

> **Shared Intermediate Condition:** Two words may be said to be core-related if they both satisfy the Relation and Containing conditions with a third concept.

This condition accounts for the close relationship among words that make reference to common concepts in their definitions. In examples (4) through (7), the following pairs of words are core-related because they satisfy this condition: *give* and *get*, *exit* and *enter*, and finally *open* and *close*.

The fundamental phenomenon captured by the Relation, Containing, and Shared Intermediate conditions is the idea that a sharing of important core concepts underlies the perceived relationships among the words in (4) through (7). In each case, the concept underlying the word makes direct reference to the core, either as the definition of the word itself or as an important component.

3.3.2. Metaphor and Core Structures

Now consider the following metaphorical uses of the same words considered in examples (4) through (7).

(8a) John *gave* Mary a cold.
(8b) Mary *has* a cold.
(8c) Mary *got* a cold from John.

(9a) Mary *entered* emacs from the shell.
(9b) John is *in* the editor.
(9c) Mary *exited* the editor without saving her files.

(10a) You can *kill* a process by typing kill process-id to the shell.
(10b) My emacs just *died*.

(11a) You have to *open* a file before you can access it.
(11b) The file is *open* for read access.
(11c) The file will not be written until it is *closed.*

The phenomenon illustrated by these examples is that the conventional metaphorical use of these words partially preserves the core-relationships established among the words in the source domain. This preservation of core-relations is captured by the following principle.

> *Metaphor Preservation Principle:* The metaphorical use of core-related words in a given target domain will hierarchically preserve the core-relationships established among the corresponding words in the source domain.

This section will illustrate the nature of this metaphorical preservation of core-relationships. In particular, each of the three core-relatedness conditions will be examined to see how it is preserved metaphorically.

Consider the metaphorical uses of *give* and *have* in (8). The Relation and Containing Conditions are satisfied between the target concepts of these metaphors in ways that directly reflect the core-relationship between the source concepts. The core-relationship established between *give* and *have* in the source domain was based upon the direct result relationship between the action of giving and the resulting state of having. This direct relationship satisfied the Relation Condition. The Containing Condition was satisfied because all the sub-parts of the having concept played roles in giving. The target concepts in (8) exhibit a similar structure. The target concept underlying the use of *give* in (8a) is an action that results in a person being infected with a cold. This resulting state of infection also plays the role of the target in the use of *have* in (8b). In a similar fashion, the sub-concepts of an infected-state play an integral role in the definition of the action of causing an infection. The Relation and Containing Conditions, therefore, hold between the target concepts of *give* and *have* in (8), in the same way that they hold between the source concepts.

Finally, consider the relationship between the metaphorical use of *give* and *get* in (8). In the source domain, these were core-related because they satisfied the Shared-Intermediate Condition. This condition was satisfied by virtue of their common connection to the concept having. The metaphorical uses of *give* and *get* are also core-related because of the Shared-Intermediate Condition. They share the common core concept of a person being in an infected-state. This infected-state is the metaphorical equivalent of the core possession concept in the source domain.

3.3.3. Structure of Extended Metaphors

The previous section discussed how the metaphorical use of core-related words in a target domain will preserve the core-relationships established among the corresponding words in the source domain. This section will show how this preservation of core-relationships is directly reflected in the representation of conventional metaphors.

In the discussion of (8), it was shown that the concepts of possession and infection formed the core source and target concepts in a set of related uses involving the words *give*, *get*, and *have*. The Infection-As-Possession metaphor is the explicit representation of the metaphor that associates these core source and target concepts. In the same way that possession and infection lie at the core of a set of source and target concepts, the Infection-As-Possession metaphor forms the core of a set of related metaphors. In particular, it lies at the core of the metaphorical uses of *give*, *have*, and *get* from (8). The explicit representations of the Infecting-As-Giving and Become-Infected-As-Getting metaphors make use of the Infection-As-Possession metaphor in a way that mirrors the core structuring in the source and target domains.

This common shared metaphor will be referred to as a *core metaphor**. Correspondingly, an *extended metaphor* is a metaphor that includes all the associations of a core metaphor and adds new associations that coherently extend the core metaphor. Table 1 gives some examples of core metaphors and various extensions to them.

For example, consider the core metaphor Process-As-Living-Thing. This core metaphor is the basis for the extended metaphors Terminating-As-Killing and Termination-As-Death. In general, the Process-As-Living-Thing metaphor is used to structure the actions that processes perform and the actions that are performed upon them. The Termination-As-Death metaphor structures the target concept of termination in terms of the death of a living thing. The corresponding Terminating-As-Killing metaphor allows the actions that cause the termination to be viewed as a killing (an action that causes a death). The Process-As-Enclosure metaphor is a distinct core metaphor that also structures some kinds of processes. It primarily structures the target concepts of actively using a process, starting to use a process, and finishing the use, in terms of entering, being enclosed within, and exiting. In each of these cases, the extended-metaphor contains the core-metaphor and extends it with the addition of further associations.

The combination of the Core-Relation Conditions with the Metaphor Preservation Principle gives rise to a corresponding set of well-formedness conditions for extended metaphors. More precisely, a core metaphor extends to another metaphor if it meets the following three requirements.

Subset Requirement: The extended sense must contain all the core associations as a subset of its associations.

Closeness Requirement: The source and target concepts of the additional associations must be "closely connected" to the source and target concepts specified by the core associations.

* The term core-metaphor, and the underlying idea, resulted from long discussions with Robert Wilensky.

Core - Infection-As-Possession → *Mary has a cold.*

Extensions

Becoming-Infected-As-Getting → *John got his cold from Mary.*

Causing-Infection-As-Transferring → *Mary gave John a cold.*

Core - Process-As-Living-Thing

Extensions

Terminating-As-Killing → *How can I kill a process?*

Termination-As-Death → *My emacs just died.*

Core - Process-As-Enclosure

Extensions

Using-Process-As-Enclosed-State → *You can edit files when you are in the editor.*

Invoking-Process-As-Entering → *You can get into lisp by typing lisp to the shell.*

Uninvoking-Process-As-Exiting → *How can I get out of lisp?*

Table 1: Core and Extended Metaphors

Coherence Requirement: The new source and target concepts must be connected to the core source and targets in a coherent manner.

In effect, these requirements force the representation of core-related metaphors to reflect the core structure inherent in the source and target domains.

The Subset Requirement ensures that the definition of the extended metaphor includes all the associations of the core metaphor. This requirement arises from the fact that it is the entire set of associations in the core metaphor that gives the metaphor its meaning. A metaphor is not an extension to another if they merely share some of the same maps.

The Closeness Requirement is a constraint on the additional maps that are added by the extended metaphor. The source and target concepts of the additional maps must be closely related to the source and target concepts in the core-metaphor. This requirement basically says that the additional concepts must be closely tied semantically to the core concepts. The details of how this requirement will be enforced are discussed in terms of the details of the representation in the next chapter.

The Closeness Requirement says that the new source and targets must be conceptually connected to the ones originally specified by the core metaphor. The Coherence Requirement further specifies that the relationships that hold between the old and new concepts in the source and target domains must be coherent with one another. Two relationships from different domains may be said to be coherent if they can be shown to be "fundamentally the same" at some level of abstraction. The Coherence Requirement follows directly from the Metaphor Preservation Principle, which predicts that extended metaphors will preserve relations from the source to target domain.

Consider, as a concrete example, the Causing-Infection-As-Transferring extension to the Infection-As-Possession metaphor. The core metaphor contains the associations that an infection is an object and that possession is a state of being infected. The extended metaphor satisfies the Subset Requirement since it contains the following core associations: the infection is the transferred object and the resulting state of infection is the possession resulting from the giving.

The Closeness Requirement is fulfilled because the source and target concepts added by the extended metaphor are conceptually close to the original source and targets of the core metaphor. In the source, the additional metaphorical concept is a transfer of possession. This is an action that results in the original source concept of possession. The core and extended source concepts are therefore related by a single relation. The target domain extends in a similar fashion. The original target concept involves an infected-state. The additional concept is an action that results in an infected-state. Therefore, both the source and targets of the extended metaphor are connected to the original metaphor by being actions or events that result in a state that is a part of the core metaphor.

The Coherence Requirement is also satisfied because the source and target concepts are extended in a similar fashion. In both the source and target domains, the extended concepts are related to the core concepts by a result relation. While the exact details of these result relationships differ, they are at a higher level of abstraction the same relation. The Coherence Requirement is therefore satisfied.

The Core-Relation Conditions, Metaphor Preservation Principle, and Extended Metaphor Requirements are criteria derived from an observation of typical kinds of conventional metaphors in the language. They are intended to perform two tasks. The first is to help capture the conceptual structure underlying the system of conventional metaphors in the language. The second is to provide a formal basis upon which programs that interpret and learn conventional metaphors can be built. Chapters 5 through 9 will show how programs that exploit the constraints imposed by these criteria can successfully perform these interpretation and acquisition tasks.

3.3.4. Representational Requirements of Extended Metaphors

We are now in a position to consider the implications of these extended metaphors for the representation. The most important requirement implied by the core-structuring of extended metaphors is that the representation must facilitate the sharing of component metaphorical associations among related senses. In the previous section, it was suggested that the metaphors themselves must be represented at the level of complex concepts. The requirement raised here for sharing implies that the component associations themselves be raised to the level of independent concepts. These independent concepts can then be shared by the various related senses that have common component associations.

This sharing, of what in a sense are reified associations, must be augmented by another facility. The related senses discussed in this section actually contain associations that are of the same kind but are not identical. The associations in the extended senses are usually specialized versions of the ones inherited from the core sense. For example, the core metaphor underlying Process-As-Living-Thing merely specifies that the process be viewed as a living thing. In example (4), this association is further specialized to be one that views a terminated process as a living thing being killed. Similarly, in the extended Transferring-As-Infecting metaphor, the original core association merely specified that the infection is viewed as a possession. In the extended metaphor, this association is further specialized to be a possession that is capable of being transferred. The other extended examples all contain similar examples of core associations being further specialized in the extended senses.

This type of sharing suggests a representation in which concepts can inherit and specialize attributes. In the case of component metaphorical associations, it should be possible to inherit and specialize the source and target restrictions posted in the core metaphors. In extended metaphors, the specializations will always involve adding restrictions that specify the role that the concept plays in the extended sense. In example (4), the added restrictions specify that the process plays the role of the victim of the killing.

The final requirement suggested by the extended metaphor phenomenon arises from the proposed extension requirements. In order to make judgments based on these rules it must be possible to answer questions concerning conceptual closeness and types of inter-connection within source and target domains. In other words, it must be possible to determine that two source or target concepts are closely connected and what the exact nature of the connection is.

The discussion up until now has been concerned with the representational requirements of the metaphors themselves. Of course, the representation of the source and targets themselves is critical to the representation of the metaphors. It has already been suggested that individual metaphors and their component associations should have the status of full-fledged concepts. The current needs suggest that as much as possible the non-metaphorical concepts that serve as the source and targets of metaphors should also have this object-oriented flavor.

In particular, to make judgments based upon the Relation and Containing Conditions it must be possible to answer questions about both the fine-grained structure within concepts and the relations that link separate concepts. Consider the giving concept discussed above. During the course of the discussion of the core-relationships of this concept and its metaphorical uses, it was necessary to make assertions about the following concepts: the giver, recipient, given-object, and the relationship of this concept to others like having and getting. These concepts and relations should exist at the level of full-fledged objects in order to easily make judgments about them. In particular, the Extension Requirements necessitate that it be possible to make judgments of type, closeness, and identity among these concepts. Chapter 4 will discuss how this representation is achieved.

3.4. Metaphor Hierarchies

As mentioned above, the representation of metaphoric associations must be able to accommodate the use of essentially the same associations in extended metaphors. It has been shown that, in these extended senses, the shared associations must be further specialized. This section will discuss a different phenomenon that will independently suggest a hierarchical representation for component metaphorical associations. Consider the following examples.

(12) How can I kill a process?
(13) Those ideas won't bear any fruit.
(14) Inflation is eating up our savings.

Each of these metaphors consists of a distinct set of component metaphorical associations. Unlike the examples in the previous section, one would not say that these are examples of extensions from the same metaphor. The source and target concepts of these examples are too dissimilar. However, each example does involve a component association that links an abstract concept that is not alive with a kind of living thing. This structuring allows us to understand some aspect of the target domain in terms of a source concept normally associated with living things. In (12), as explained above, a process is viewed as a living thing, allowing termination to be viewed as killing. In (13), ideas are metaphorically viewed as plants, allowing the creation of new ideas from old ones to be viewed as producing fruit. Finally in (14), inflation is viewed as something that can eat, allowing the relative reduction in savings by inflation to be viewed as a kind of consumption.

Each of these underlying associations is different in the specifics of the particular source and target concepts used. However, they all associate an abstract non-living thing with a specific kind of living thing. This suggests the idea that these associations should be represented at the level of full-fledged objects. This will allow them to be included in abstraction hierarchies. Abstraction hierarchies can efficiently handle the kind of sharing that these associations seem to exhibit. The commonalities among the various associations are captured by creating a more abstract association between the proper abstract source and target concepts. In this case, an association linking a concept representing living things with another concept representing non-living things is created. The particular associations underlying (12) through (14) can then inherit and specialize the source and target components of the more abstract association. This use of hierarchies is the fundamental mechanism that will be used to account for subtle similarities and differences among metaphors.

3.5. Similarity Among Metaphors

The previous section discussed some of the similarities among the building block associations from which metaphors are constructed and how these similarities might be captured by a hierarchical representation. This section addresses the issue of overall similarity among metaphors. In particular, it will show that by elevating metaphors and their component associations to the level of concepts, a wide range of phenomena generally referred to as similarity as can be accounted for.

Consider the following examples.

(15) Mary gave John a cold.
(16) Mary gave John the flu.

On the surface these seem like two distinct highly similar metaphors. In particular, while they have the same source concepts, they have different target concepts. However, it would not be correct to represent these as two separate metaphors, even though they have different targets. The two target concepts are both instances of the more general category corresponding to something like communicable disease.

A single metaphor with a target concept represented at the level of communicable disease best captures Examples (15) and (16). The specific target concepts, infecting with a cold and infecting with the flu, are represented as members of a more general infecting with a communicable-disease category through the use of an abstraction hierarchy. This representation allows us to represent the metaphor at a correct level of abstraction and still arrive at an appropriately specific interpretation with little effort.

Now consider the following examples:

(17) How can I kill a process?
(18) John killed the conversation by walking in the room.

While one would not ordinarily say that conversations and processes are similar, they do share some significant elements that are referred to by this metaphor. The shared elements focussed on in these examples have to do with these concepts being events that occur over time that can be terminated. These shared elements are inherited from the fairly abstract concept of event or activity.

In a sense, these metaphors are creating a similarity by focussing on those aspects of the concepts that are shared. These common features are represented as explicit relations and concepts that the two target concepts have in common. As with the previous examples, the representation must be flexible enough to permit the metaphor to be represented at the correct level in the hierarchy with respect to these common elements. In this case, the metaphor should express the notion that killing can be used to view an abstract event corresponding to termination. This abstract concept of termination must be related hierarchically to the specific concepts of terminating conversations and computer processes.

In the following examples, a more distant kind of similarity is considered.

(19) Mary gave John an idea.
(20) Mary gave John a cold.

In these examples, the target concepts are completely distinct. Independent of any metaphorical associations, ideas and colds are not significantly related conceptually. The process of coming to have an idea has little in common with the concept of becoming infected with a disease. Nevertheless, the metaphors share the basic structure of viewing a set of actions by one participant leading to a change in another participant as a transfer to that participant. In addition, in both cases, an abstract concept is being viewed as an object that can be transferred and possessed. This abstract metaphor corresponds to the Transfer-As-Act-Upon metaphor described in Jacobs (1986).

An important trade-off issue is raised by the existence of these highly abstract metaphors. In particular, consider the situation where the Transfer-As-Act-Upon metaphor is applied directly in an attempt to understand (20). The result of this is an abstract target concept representing the idea that Mary acted upon John in a way in which a cold played some role. This is clearly not an adequate interpretation. Some further inference process is needed to replace this concept with a more appropriate specific concept indicating that an action took place that resulted in Mary's being responsible for John becoming infected. The problem is that this further specialization inference may be computationally very expensive to perform. In general, the more abstract the representation of the metaphor, the more expensive it is to arrive at an appropriately specific target representation. Therefore, the goal of capturing these abstract generalizations seems at odds with the goal of providing appropriately useful knowledge.

Consider the alternative situation where the more specific Infecting-As-Giving metaphor, suggested for examples (15) and (16), is applied. The use of this more specific metaphor leads to an appropriate target concept much more directly. The trade-off therefore arises from the representation and use of these abstract metaphors. A system that merely represents specific uses does not capture any generalizations, and therefore cannot easily accommodate new examples. On the other hand, the actual use of highly abstract metaphors is more computationally expensive. Braverman (1988) discusses the general trade-off issues implied by this generality versus operationality issue.

The solution taken in this thesis is to represent both specific and abstract metaphors in an abstraction hierarchy. The most specific metaphor that can be applied is the one selected for use during normal processing. The existence of higher level abstract metaphors, in the same hierarchy with more specific instances, facilitates the interpretation of new uses. Chapter 7 will illustrate how these abstract metaphors can be used in conjunction with more specific metaphors to learn new uses.

3.6. Challenges to Metaphorical Regularity

A natural issue to consider in light of the regularities described here is that we must be able to explain those uses that seem like they should plausibly work, given a core metaphor and a proposed target, but clearly do not. These missing metaphors might pose a significant problem for a theory that asserts that conventional metaphors are not simply random idioms, but rather arise from systematic conceptual structures.

Consider the following examples.

(21) How can I kill a process?
(22) *How can I give birth to a process?
(23) *How can I slay a process?

(24) John gave Mary a cold.
(25) *John donated his cold to Mary
(26) *John took Mary a cold.

(27) Mary caught a cold from John.
(28) *John threw Mary his cold.

Within each set of examples, the first example results from a valid use of a previously discussed conventional metaphor. The remaining examples, within each set, seem plausible given the related known metaphors but simply don't work. These kind of examples undermine the assertion that metaphors are the result of underlying conceptual structures and are not arbitrary uses.

Consider again the core-metaphor of a Process-As-Living-Thing metaphors that underlies (21). It was shown in Section 3.3.3 that the actions and events that lead to the termination of processes can be viewed as killing and dying. It would seem plausible to suggest that the actions by a user leading to the creation of a process might be structured in terms of birth events. It seems, however, that this is not the case. In particular, (22) does not seem to work very well to convey the idea of creating a process.

It can, however, be shown that the reason for these failures is not inconsistent with the regularities described in this chapter. In particular, this use fails to meet some of the extension requirements set forth in Section 3.3.3. Consider the actual details underlying (22). The target concept of creating a process seems to be cognized as a simple action by a user that results in the creation of a process. In order for there to be a coherent extension from the core metaphor, there would have to be a corresponding simple action by an actor that results in the creation of a living-thing. The complex concept of a birth-event does not meet this requirement. There is no single action causing the final result. Rather, it is a complex series of events involving several participants. A birth metaphor would, therefore, have to map the simple target action to a set of source events.

Another serious mismatch between the create concept and the birth concept involves the restrictions on the creator and the mother (the participant giving birth in the birth event). The birth event has the restriction that the created thing and the mother are the same type of entity. This does not correspond to the target domain adequately. The general concept of creation that dominated process creation does not assume or require any type of connection between the thing created and the creator. The process is thought of as being created by the user not by another process. The net result is that the connection between the core metaphor and the target concept of the creation of a process does not match the connection from the core to the source concept of birth in a consistent enough manner to allow this use.

Consider, however, the fact that conventional UNIX language does permit the use of *spawn a process* to refer to the creation of a process by some other process. In this case, the semantics of the source concept of *spawn* does not involve the mismatches with the target concepts that occurred with (22). In particular, the concepts underlying *spawn* seem to be simpler then the concepts underlying *give birth to*. Spawn focusses only on the final act of releasing the young and not to the entire process. In addition, the semantics of the target concepts in this use does not have the type mismatch noted above between the created thing and the creator. It is used here to refer to the creation of a process by another process.

A similar analysis rules out the prediction of *throw* in (28). This use seems to be predicted by (27). As initial analysis would assert that since *catch a cold* conventionally means to become infected then *throw a cold* should mean to infect. This analysis fails because of the simplistic analysis of the source domain of the known metaphor. In this case, the incorrect assumption would seem to be that this source use of *catch* is the sense that takes part in the catch/throw pairing. It seems instead that this use involves one of the many senses of *catch* that does not relate to *throw* at all. In particular, the non-conventional nature of *catch a cold* seems to imply that the most likely use is the sense underlying (29).

(29) John caught his sleeve on a nail.

This sense emphasizes the notion that the catching is an event that happens to someone without any effort or intention on the part of the catcher*.

There is a common thread running through each of these analyses. In each case, minor meaning differences among words, and among senses of the same word, have important implications for the metaphorical uses of those words. In particular, these subtle differences may prevent the extension of a core metaphor to what seems like a reasonable use. The conceptual metaphors presented in this chapter are, therefore, regular and productive but they have very specific semantic conditions for their use. The missing metaphors in Examples (21) through (28), therefore, do not pose a challenge to the theory of metaphor presented here. Rather, the challenge they pose is to work out the subtle semantic inter-relationships among the meanings of the words in the language.

3.7. Motivation

The idea that subtle similarities and differences among the facts that are being modeled should be reflected in the representation is a fundamental one. It motivates most of the knowledge structures proposed in this chapter. This approach has two motivations. The first motivation follows from one of the basic motivations underlying the development of KODIAK. Wilensky's *Cognitive Correspondence Principle* (Wilensky 1987) states that:

> *A particular representation for a particular item must be supported by its correspondence to how that item is cognized.*

We take the empirical data provided by particular conventional metaphors to be evidence for how the underlying concepts are cognized, and are therefore represented. While the principle only addresses the representation of particular concepts, it follows that the similarities and differences among sets of facts must be represented as well.

The second motivation for representing these similarities and differences is more immediate to this thesis. Chapter 6 will show how the explicit representation of these facts enables certain kinds of learning behavior.

3.8. Summary

This chapter provided an analysis of the characteristics of conventional metaphors that need to be captured by a knowledge representation. This analysis of individual metaphors and of sets of related metaphors produced the following four representational requirements.

* The analysis of this sense of *catch* was suggested by Robert Wilensky.

Fundamental Representational Requirement: Conventional metaphors must be explicitly represented as coherent sets of associations between source and target concepts.

Extended Metaphor Requirement: The representation of conventional metaphors must capture the fundamental phenomenon that a sharing of important core concepts underlies the perceived relationships among many separate metaphors.

Sharing Requirement: The representation of the component associations of metaphors must facilitate the hierarchical sharing of these component parts among distinct metaphors.

Similarity Requirement: The representation of conventional metaphors must capture the similarity relationships among metaphors.

The Fundamental Representational Requirement addresses the need to account for the fact that these metaphors are conventional parts of the language. These metaphors can be accounted for by a knowledge-base containing sets of associations between source and target domains that are conventionally related. These associations must be specific enough to account for the particular details of individual metaphors.

The remaining requirements arise from the desire to capture the systematicities among the set of metaphors in the language. These systematicities were explored by considering sets of seemingly related metaphors. In each case, the phenomenon was characterized and suggestions were made as to an appropriate representation. It was suggested that both extensions to core metaphors and similarities among metaphors can be accounted for by a representation that permits the specialized sharing of component elements. In such a representation, both metaphors and their component associations are elevated to the status of concepts. A knowledge representation system that incorporates a system of hierarchical inheritance can account for this type of sharing. The next chapter will describe the details of the representation language that is used to capture the requirements posed by this analysis of conventional metaphor.

Chapter 4

Knowledge Representation

4.1. Introduction

Chapter 3 provided an analysis of conventional metaphor and provided some requirements for the representation of metaphoric knowledge. This chapter describes exactly how this task is accomplished. I first briefly describe KODIAK, which is the knowledge representation language used to accomplish this task. I then review the aspects of conventional metaphor that need to be captured, and show how this is accomplished using KODIAK.

4.2. KODIAK

KODIAK is a knowledge representation language developed by Robert Wilensky and various members of the BAIR group at Berkeley. The motivations for its development and its theoretical underpinnings are best described in Wilensky (1986). The actual implementation described here is a modified version of the one developed by Norvig (1986) for the FAUSTUS text inferencing system. The description of KODIAK provided here will be brief, introducing only those ideas and notations needed in order to follow the rest of the thesis*. More details will be introduced along the way as necessary.

KODIAK is best seen as an extended semantic network language in the tradition of KL-ONE and its variants. Facts in KODIAK are represented as nodes connected together with primitive links. The language provides three types of nodes and eight primitive kinds of links. Table 1 from Norvig (1986) gives a brief description of each node and link type.

* KODIAK as a language and as a theory is in an almost constant state of flux. Therefore the details described here differ is detail but not in spirit from those described in Jacobs (1985), Wilensky (1986), and Norvig (1986).

Absolutes - concepts, e.g. person, action, idea
Relations - relations between concepts, e.g actor-of-acting
Aspectuals - arguments of relations, e.g. actor

Dominate - a concept is a sub-class of another class
Instance - a concept is an instance of a class
View - a concept can be viewed as another concept
Constrain - fillers of an aspectual must be of some class
Argument - associates aspectuals with a relation
Fill - an aspectual refers to some instance
Equate - two concepts are co-referential
Differ - two concepts may not be co-referential

Table 1: Primitives of KODIAK

The three types of nodes or objects are *absolutes*, *relations* and *aspectuals*. Most concepts that intuitively are thought of as standing on their own are represented as absolutes. This generally includes notions like physical objects, people, ideas, actions, events, and sets. Examples of absolutes that we will see are living-thing, person, action, giving, and having. The major concepts underlying the meanings of most nouns and verbs will turn out to be absolutes.

Relations are concepts that are used to associate two objects. It is important to distinguish between relations and primitive links. A relation is a complex concept composed of nodes and links that is used to associate two other concepts. A primitive link is used to connect two nodes in the system. While there are only eight links known to the interpreter, the knowledge base may contain an arbitrary number of user-defined relations. The links in most simple semantic network systems would become relations in KODIAK.

The final type of object is an aspectual. Each relation has two associated aspectuals that specify the range and domain of the relation. Consider the part-whole relation intended to associate concepts that are in a part-whole relationship. This relation has two related aspectuals corresponding to the part and the whole.

Figure 1 shows the representation of the part-whole relation and two of the eight primitive links that will be discussed. The horizontal line in the center labeled `part-whole` represents the relation. The two circles represent the aspectuals `part` and `whole`. The boxes represent absolutes that serve as the range and domain of the relation. The diagrams in this thesis will follow the following conventions: boxes are absolutes, circles are aspectuals, labeled horizontal lines are relations, primitive links are shown as arcs with single character labels.

The arcs labeled *a* between the relation and its aspectuals are primitive links called

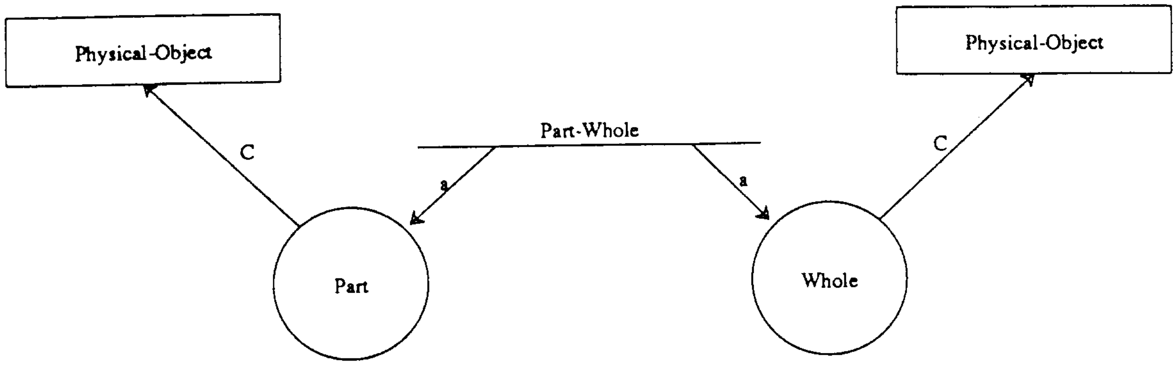

Figure 1: Part-Whole Relation

arguments. They are used to connect a relation to its attendant aspectuals. The links labeled C are used to connect an aspectual to the category that *constrains* that aspectual. This basically says that this aspectual is constrained to be a member of the constraining category.

Figure 2 illustrates the use of inheritance in KODIAK. In this diagram, the `has-table-top` relation is shown to be a kind of `part-whole` relation. The various vertical lines labeled *D* stand for *dominate* links. A dominate link specifies that the lower concept is a sub-class of the concept pointed at. In this example, we see that the `has-table-part` relation and its aspectuals are dominated by the parent `part-whole` relation. In addition, the absolutes `table` and `table-top` are kinds of `physical-objects`. Dominate links are therefore used to specify inheritance for absolutes, relations, and aspectuals. Dominate links are transitive; concepts inherit relations from their direct parent and all of the parent's ancestors.

Another kind of inheritance link is the *I*, or *instance*, link. The I link is used to denote the fact that a concept is an instance of a category. Individual tokens of absolutes, relations and aspectuals are linked to their immediate parent via an I link. While the dominate link may be considered a subset relation, the instance link is an element-of relation. Another link, the F or *filler* link, goes along with this instance notion. The aspectuals of instances of relations are connected via F links to the instances that constrain them. The filler of an aspectual must satisfy all the constraints that the aspectual inherits from its parents.

The =, or *equate*, link is used to assert that two concepts are co-referential. The equate link can hold either between absolutes or aspectuals. Corresponding to the equate link is the *differs* link that states that the two concepts may not be equated.

The remaining link, the V, or *view*, link is used to assert that the one concept may in certain circumstances be considered as another. A V link is more complex than the

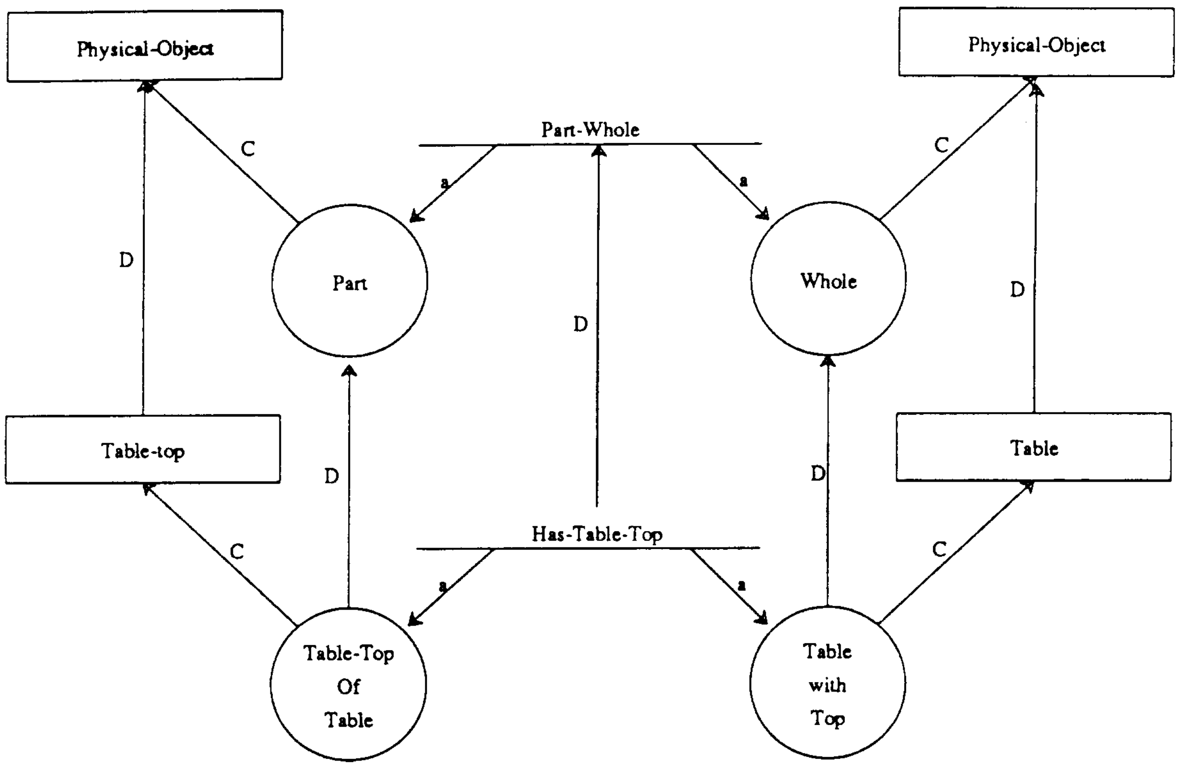

Figure 2: Has-Table-Top Relation

links discussed so far and it will be discussed in greater detail in the following sections.

The final concept introduced here is a notational one developed by Norvig (1986). This is the called the *slot* notation. It is frequently the case that when a relation between two absolutes is being shown, it is in terms of one of the absolutes relating to the aspectual farther away from it. Consider Figure 3. This diagram shows a concept called a `killing` which is in a `has-kill-victim` relation to the category living thing. The related aspectuals are the `killing-with-victim` and the `kill-victim`. It will frequently be the case that we are interested in only the role of the `kill-victim`. The slot notation handles this situation. The lower part of Figure 3 shows the slot abbreviation of the `has-kill-victim` relation. The relation and the aspectual closest to the concept with the slot are replaced by an S link. The remaining aspectual and its constrainer remain the same. Remember that this is only a notational convenience to reduce the complexity of the diagrams. The actual underlying representation remains in terms of relations, aspectuals, and argument links.

A linear lisp-like notation will also occasionally be used to describe KODIAK structures. These will usually be used when illustrating actual output from various programs manipulating KODIAK structures. The `Kill-Victim` facts represented in Figure

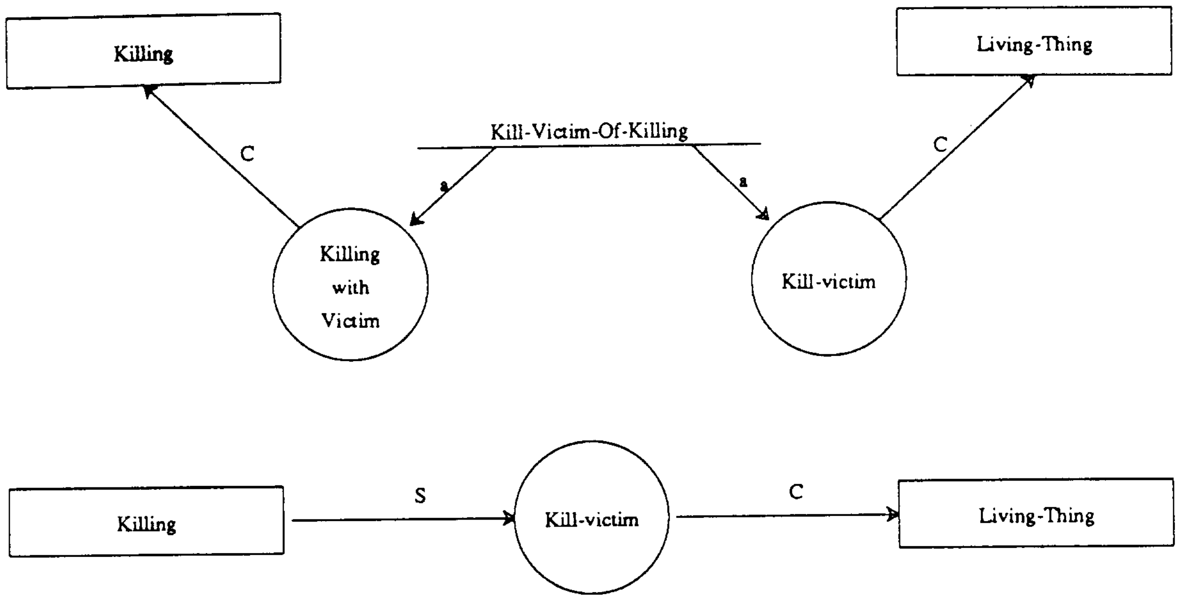

Figure 3: Slot Notation

3 are expressed with the following notation.

```
(A Killing (↑ Action)
     (kill-victim (↑ patient) (A Living-Thing (↑ Thing))))
```

This notation indicates that a `Killing` is dominated by `Action`. It also has a slot called `kill-victim`, which is dominated by `patient`, and is constrained to be a `Living-Thing`.

4.2.1. Structured Associations

The dominate, view, and instance links, described above, taken together are a primitive mechanism for providing inheritance. There is, however, a major problem with using them to provide a coherent inheritance mechanism. The problem arises from the fact that the individual links are highly local in nature; they link individual KODIAK objects. However, most meaningful concepts consist of sets of related concepts. Correspondingly, meaningful inheritance involves sets of related concepts being inherited at the same time. The clearest example of this is illustrated in the specialization of relations.

Consider again the specialization of the `part-whole` relation to the `has-table-top` relation discussed above. The coherent specialization of this concept involves the three dominate associations shown in the center of the diagram. The link that dominates the `has-table-top` relation by the `part-whole` relation presumes that the corresponding aspectuals will also be dominated. This follows from the fact that the relation cannot stand alone. The only means of specializing a relation is to further specify its aspectuals. What is required to capture this fact is a concept that ties all of these links together to indicate that they constitute a coherent set.

This coherent inheritance is accomplished in KODIAK through the creation of a concept called a *structured association**. A structured association is an absolute that consists of a set of primitive inheritance relations. The structured association is related to these primitive relations via a set of `component-association` relations. A `component-association` relation is a relation that connects an inheritance relation to a structured association. In addition, one of the components is further marked as being the primary mapping to reflect the fact that the other associations follow from this one. In the `has-table-top` relation, the primary association is the one linking the `part-whole` relation to the `has-table-top` relation. Figure 4 shows the structured association corresponding to the `has-table-top` example. There are three component associations that are D links. The primary one linking the relations.

More complex examples of structured associations will be given below in the discussion of the representation of conventional metaphors. In particular, structured associations consisting of sets of views will be used to represent individual conventional metaphors.

4.3. Conventional Metaphorical Knowledge

Chapter 3 provided an analysis of some of the characteristics exhibite[illegible]y conventional metaphors. In addition, a number of general suggestions were made for representations that might capture these characteristics. This section will review these suggestions in terms of some examples and briefly indicate the KODIAK features used to capture the suggestions. The following sections will then give the exact details of each feature.

The first requirement for the representation is to represent metaphors as concepts consisting of sets of associations between source and target concepts. Consider Example (1).

(1) How can I kill a process?

This example, from the UNIX domain, involves the conventional metaphor that to kill an

* Previous formulations of KODIAK (Wilensky 1986, Jacobs 1985 and Norvig 1986) have employed a different mechanism to accomplish the same effect as these explicit structured associations. A secondary link called a role-play was employed. The primary link was made a true relation and the rest were made dependent role-plays of that governing relation. The set of component associations used here accomplishes the same correspondence but by elevating the association to the level of a full-fledged concept a more uniform representation is achieved.

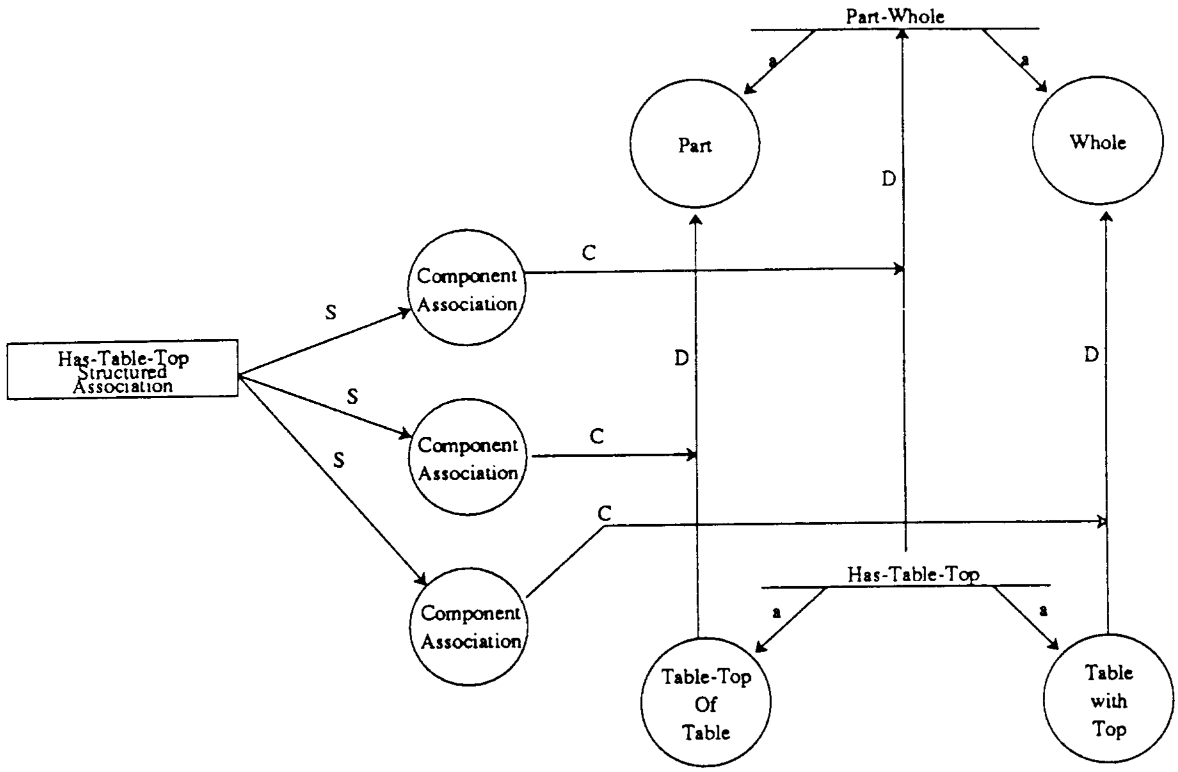

Figure 4: Has-Table-Top Structured Association

ongoing process means to terminate it. The target concepts involve computer processes and the actions that terminate them. The source concept is that of the action of causing a living thing to die. The metaphor consists of the source, target, and the set of associations linking them.

Conventional metaphors like this one are captured in KODIAK through the use of the structured association mechanism. A structured association, called a metaphor-sense, is created to represent the metaphor itself. The component associations consist of metaphoric VIEW relations called metaphor-maps. These metaphor-maps are the associations used to connect source and target concepts. Moreover, these relations are given the status of full-fledged concepts, since relations in KODIAK are concepts.

The next set of requirements for KODIAK were suggested by the extended metaphor phenomena. Consider the following examples from Chapter 3.

(2) Mary has a cold.
(3) John gave Mary a cold.

We need to be able to represent the fact that the giving metaphor underlying (3) is extended from the core possession metaphor motivating (2). This extended metaphor is accounted for in KODIAK by two mechanisms. The first is that the concepts representing these two metaphors are directly linked by a relation that specifies the giving metaphor as an extension of the possession metaphor. The second mechanism accounts for the sharing of component associations discussed in the previous chapter. As mentioned above, the component associations of the metaphor are relations and can, therefore, be specialized in the normal fashion. The extended sense, therefore, shares the associations from the core by using further specified children of the core associations.

The rest of the similarity phenomena from Chapter 3 are all accounted for by these same mechanisms. The basic approach is that metaphor-senses and metaphor-maps are full-fledged KODIAK concepts and, therefore, all the various similarities and differences can be accounted for by the basic KODIAK inheritance mechanisms.

The following sections will introduce the various mechanisms that will be used to both represent the specific details of individual metaphors and capture the necessary generalizations.

4.4. Representing Conventional Metaphors

Figure 5 shows the KODIAK representation of the source domain from (1). It states that a `killing` is a kind of `action` with a `result` that is a `death-event` which is in turn an `event`. The `kill-victim` of the `killing` is an inherited role from `action` indicating that the `kill-victim` is effected by the action. The `kill-victim` is constrained to be a `living-thing` and the `killer` must be an `animate-agent`. Finally the equate links require that the `kill-victim` must be the same as the `dier` participant of the `death-event`.

Figure 6 shows the corresponding concepts from the target domain. It states that a `terminate-process-action` is a `terminate-action` which is a kind of `action`. The `terminated-process` role is an inherited role specifying the patient of the action. The result of the action is a `terminate-process-effect` which is a kind of `terminate-event`. Finally, the `terminated-process` is equated to the `terminated-process-event` of the `terminate-process-effect`. This is analogous to the relationship between the `kill-victim` and the `dier` shown in Figure 5.

What is needed is a way of associating the appropriate source and target concepts. Such an association is realized in KODIAK by using a relation called a *metaphor-map*. A metaphor-map is a kind of VIEW relation whose aspectuals specify corresponding source and target concepts. Figure 7 shows the details of one of the metaphor-maps, the `killed-process` map that underlies (1). The `killed-process` map constrains the source aspectual to be a `kill-victim` and the target to be a `terminated-process`. The bottom part of Figure 7 shows a shorthand notation for illustrating metaphor-maps.

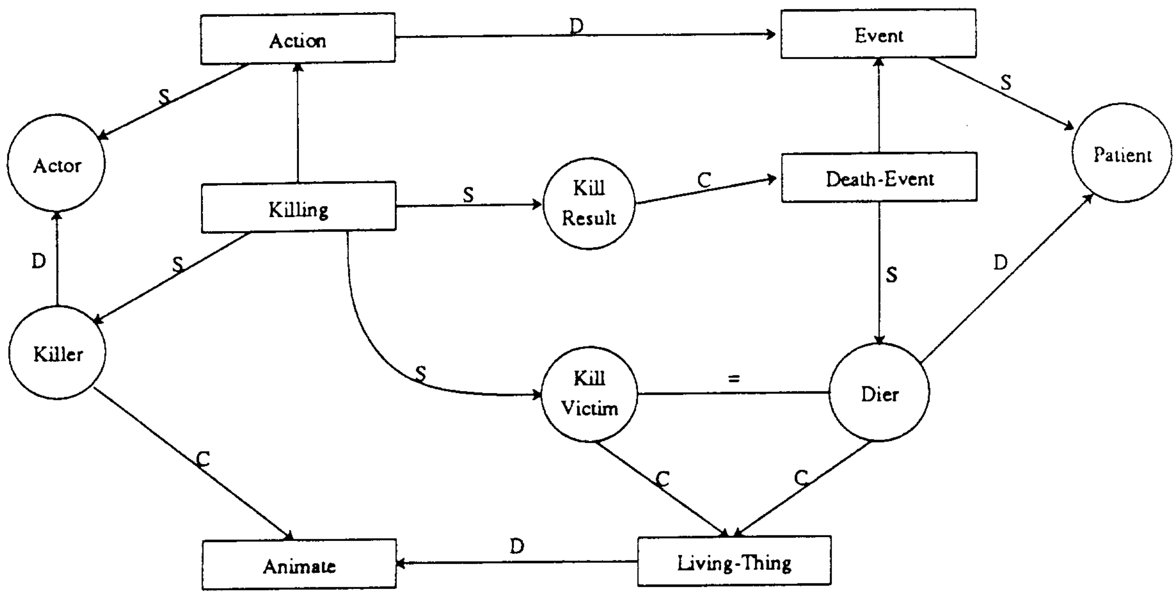

Figure 5: Killing

Metaphor maps are needed to link all the core source concepts in Figure 5 to their counterparts in the target domain. In particular, the `killing` maps to the `terminate-action`, the `kill-victim` maps to the `terminated-process`, the `killer` maps to the `actor` of the `terminate-action`, and the `result` of the `killing` maps to the `result` of the `terminating`. Figure 8 shows the complete set of maps underlying (1). It is the co-occurrence of all these maps that constitutes the conventional metaphor that terminating something can be viewed as a killing.

This co-occurrence of a set of more primitive inheritance relations is the definition of a structured association. Therefore, a kind of structured association called a metaphor-sense is introduced to capture this meaningful co-occurrence of metaphor-maps. These maps tie together concepts that are highly inter-connected in the source and target domains. In Example (1), the metaphor-maps tie a single absolute and its attendant aspectuals on the source side to an equivalent set on the target side. A metaphor-sense is, therefore, a structured association that ties together sets of component metaphor-maps that together constitute a meaningful conventional metaphor. A metaphor-sense represents a meaningful unit in the same way that the concept `killing` and its relations taken together form a meaningful unit.

A metaphor-sense can participate in an unspecified number of component-map relations. A component-map relation is a kind of component-association relation that holds between a metaphor-sense and a metaphor-map. Figure 9 shows the metaphor-sense `kill-terminate-metaphor` that ties together all the metaphor-maps underlying

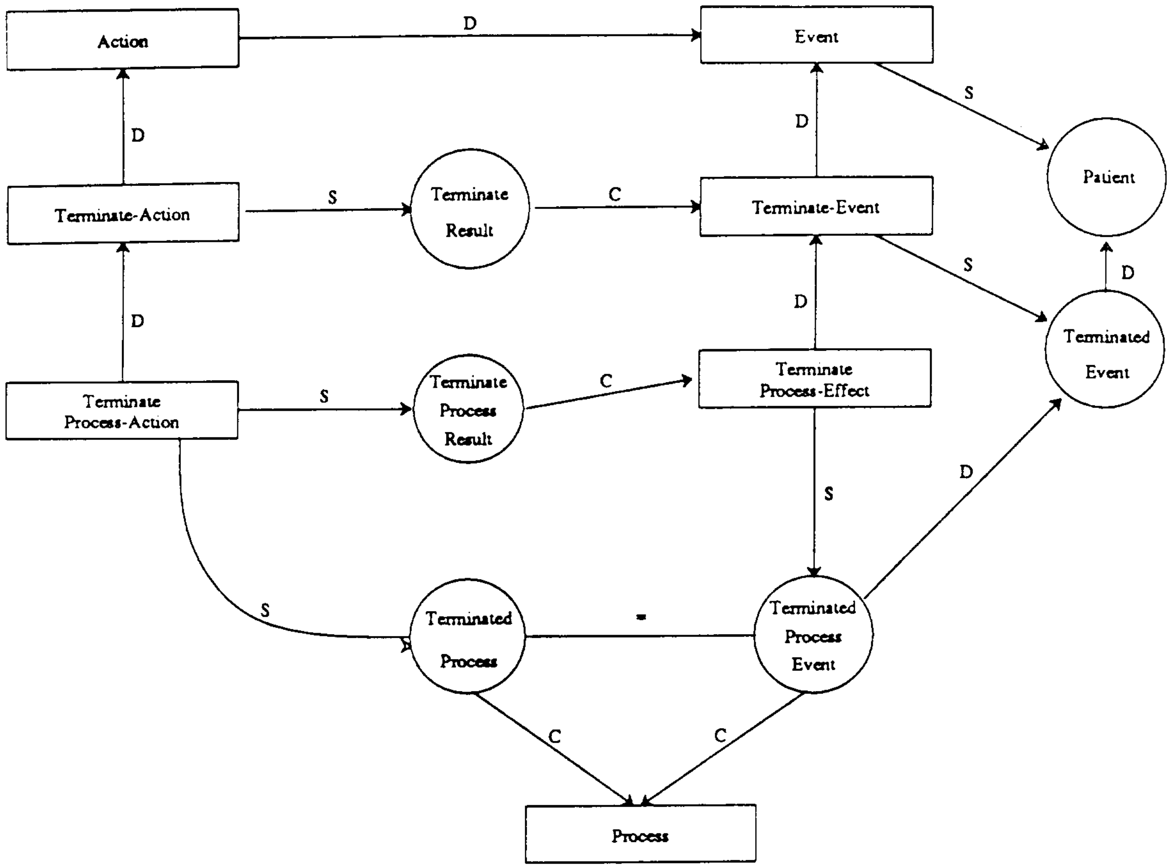

Figure 6: Terminating

Example (1). Figure 10 shows the abbreviated notation for illustrating metaphor-senses. The sense itself is represented as the box enclosing the individual maps.

To a significant extent, metaphor-senses are the minimal meaning-bearing unit of conventional metaphors. Metaphor-maps represent the building blocks out of which meaningful metaphor-senses are constructed. The metaphor-sense represents the level at which one would say that there is a conventional metaphor that to terminate something is to kill it. This level of representation will frequently correspond to a single metaphorical word sense.

4.5. Representing Extended Metaphors

Chapter 3 discussed the need to capture the relationships between separate metaphors that nevertheless seem to be related. The analysis of these extended metaphors

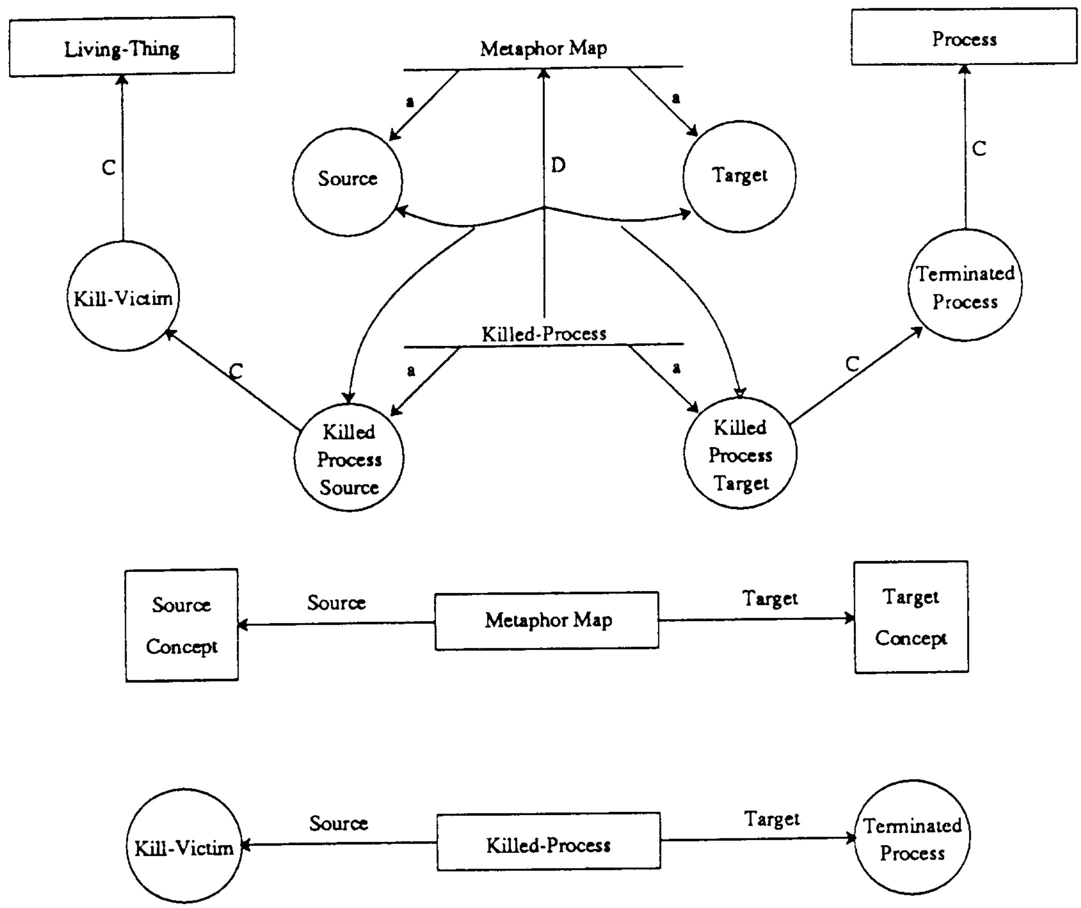

Figure 7: Metaphor Maps

resulted in a set of criteria that explained how these metaphors arise from the preservation of core-relationships in the individual source and target domains. The following sections present more detailed versions of these criteria to show how this core structuring is directly realized in KODIAK.

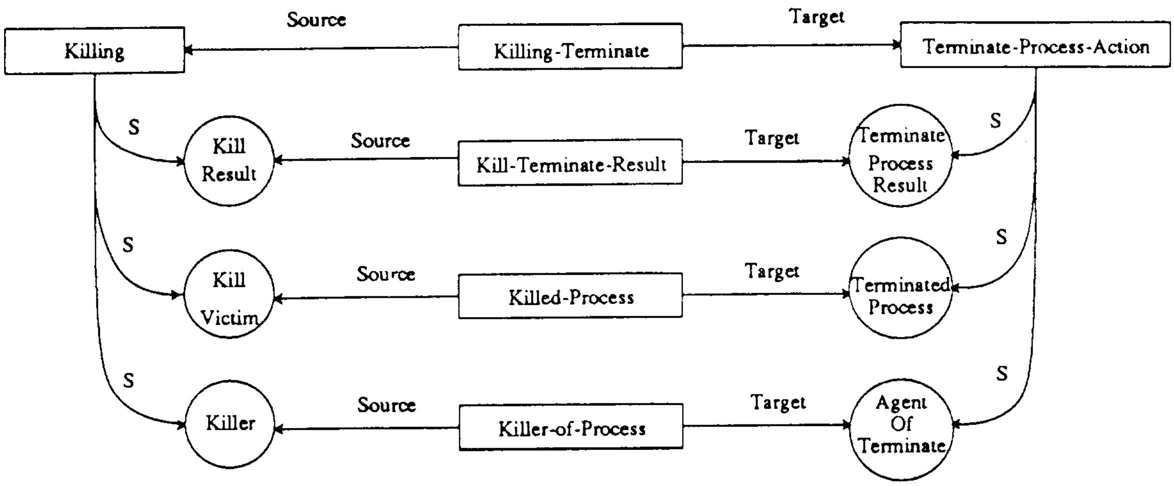

Figure 8: Kill-Terminate-Maps

4.5.1. Core Structures

Chapter 3 established three conditions that specified how concepts could be core-related. In this section, these conditions are restated in KODIAK terms. Consider the following examples.

(3a) John *gave* Mary a book.
(3b) Mary *has* a book.
(3c) Mary *got* a book from John.

Figure 11 shows the KODIAK representation of the concepts underlying the words *give*, *get* and *have* in (3). The core-relationships described in Chapter 3 are reflected in the network in a number of ways. Consider the relationships between the concepts `giving` and `having`, which underlie the words *give* and *have*. The core-relationship between these words is primarily reflected in the network by the `giving-result` concept. The `giving-result` slot indicates that the result of the `giving` action is the stative `having`. Specifically, the `giving-result` aspectual is constrained by the concept `having` and the `giving-with-result` aspectual is constrained by `giving`. Similarly, the concept `getting` is directly related to `having` by the relation underlying the `getting-result` slot.

In addition to the direct connections represented by the `giving-result` and `giving-precond` slots, there are a number of indirect structural relationships that indicate strong connections between these concepts. Consider the `givee` and `given` slots of the `giving` concept. These are equated to the `haver` and `had` slots of the `having` concept. These equate links represent the fact that the recipient of a `giving` is a `haver` and that the object given is a thing that is the `had` of some `having`.

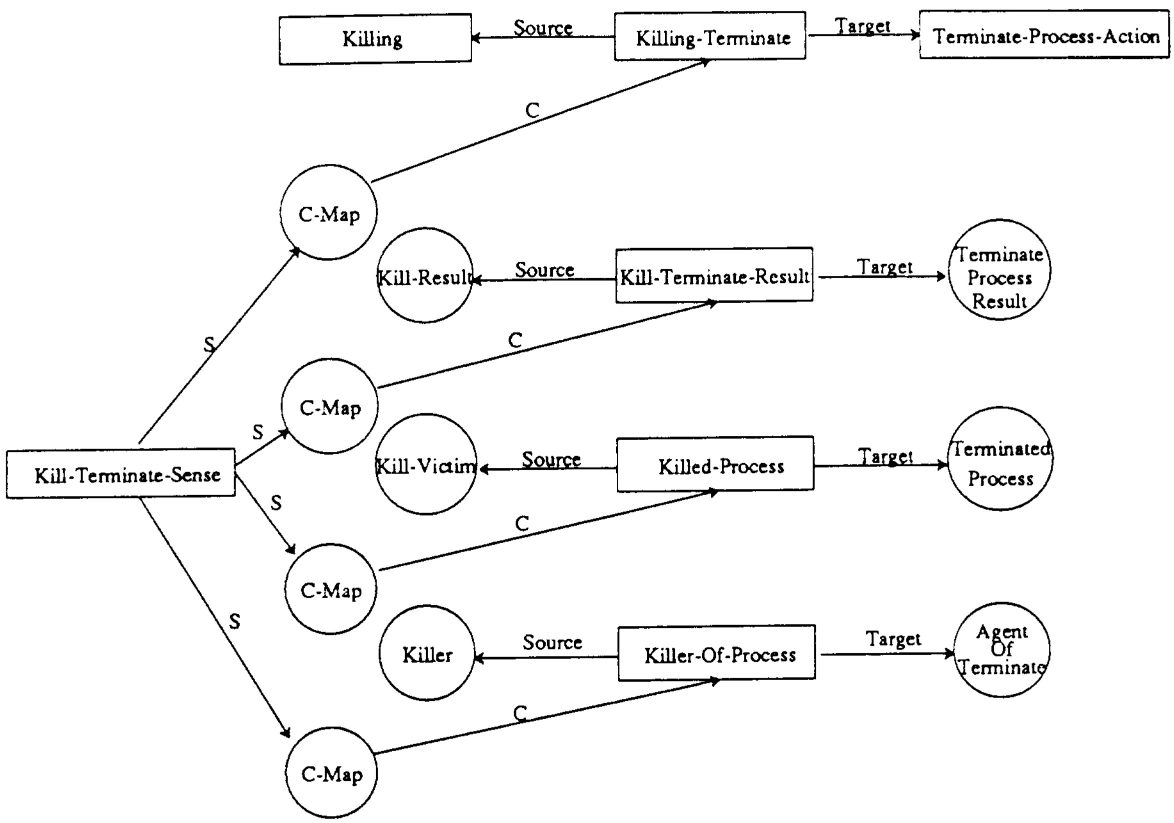

Figure 9: Kill-Terminate-Sense

The relationship between *give* and *have* illustrates the basic kind of core-relation. The words *give* and *have* are said to be core-related because the concepts underlying the words are connected directly by a single relation and all the slots of the `having` concept are equated to slots in the concept `giving`. This illustrates the situation where the definition of one word refers to, and wholly contains, the definition of another word.

The Relation and Containing Conditions, from Chapter 3, can now be restated in more specific KODIAK terms. Formally, two words can be said to be core-related if the following two conditions hold.

Relation Condition: There is at least one relation whose aspectuals are constrained by the main concepts underlying each of the words.

Containing Condition: All of the slots of the concept underlying one of the words must be equated to slots of the other concept.

These KODIAK specific conditions capture the intuition that words are core-related if the definition of one of the words makes direct reference to the definition of the other.

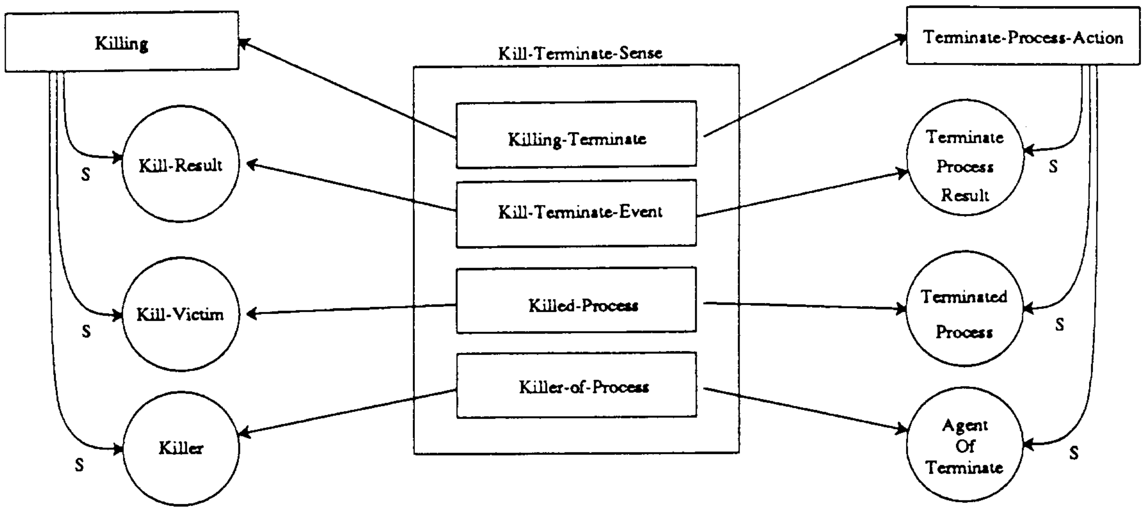

Figure 10: Kill-Terminate-Sense

The Containing condition requires that not only is the definition of the second word referenced but that it is an integral part of the definition of the first.

Now consider the relationship between the meanings of the words *give* and *get*. The concepts `giving` and `getting` are not directly related in the same way that `giving` and `having` are. Instead, they are indirectly related by virtue of the fact that they are both core-related to `having`. This satisfies the intermediate core-relation condition.

Shared Intermediate Condition: Two words may be said to be core-related if they both satisfy the Relation and Containing conditions with a third concept.

4.5.2. Metaphorical Preservation of Core Structures

Now consider the following metaphorical uses of the same words considered in (3).

(4a) John *gave* Mary a cold.
(4b) Mary *has* a cold.
(4c) Mary *got* a cold from John.

The phenomena illustrated by these examples is that the conventional metaphorical use of these words partially preserves the core-relationships established among the words in the source domain. This preservation of core-relations is captured by the Metaphor Preservation Principle introduced in Chapter 3.

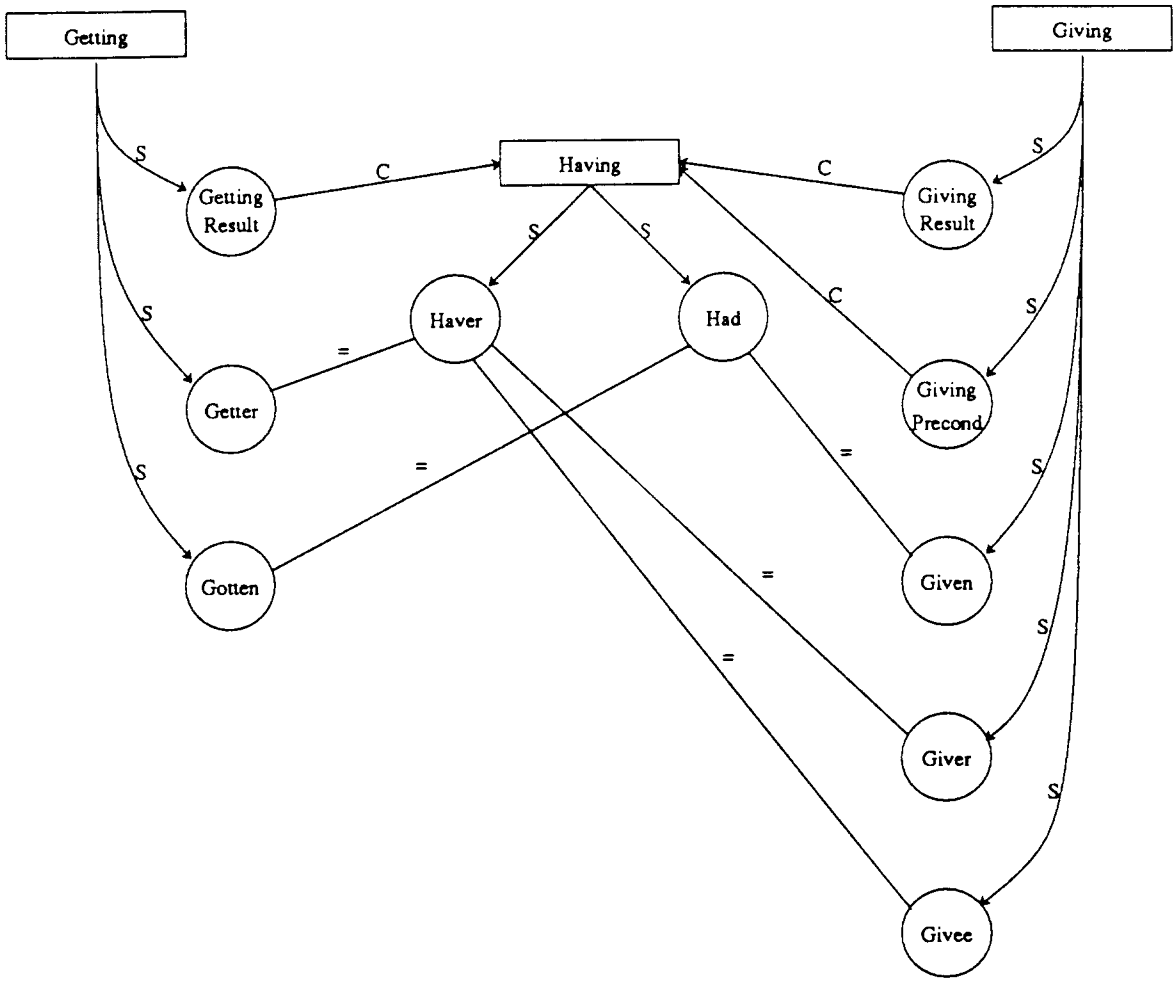

Figure 11: Giving, Getting and Having

Metaphor Preservation Principle: The metaphorical use of core-related words in a given target domain will hierarchically preserve the core-relationships established among the corresponding words in the source domain.

This section will illustrate the exact nature of this metaphorical preservation of core-relationships. In particular, each of the three conditions for core-relatedness will be examined to see how each is preserved metaphorically.

Consider the metaphorical uses of *give* and *have* in (4). The first step in determining the nature of the preservation of the core-relationships between *give* and *have* is to examine the details of the corresponding source and target concepts. The details of the target concepts of these metaphor-senses are illustrated in Figure 12.

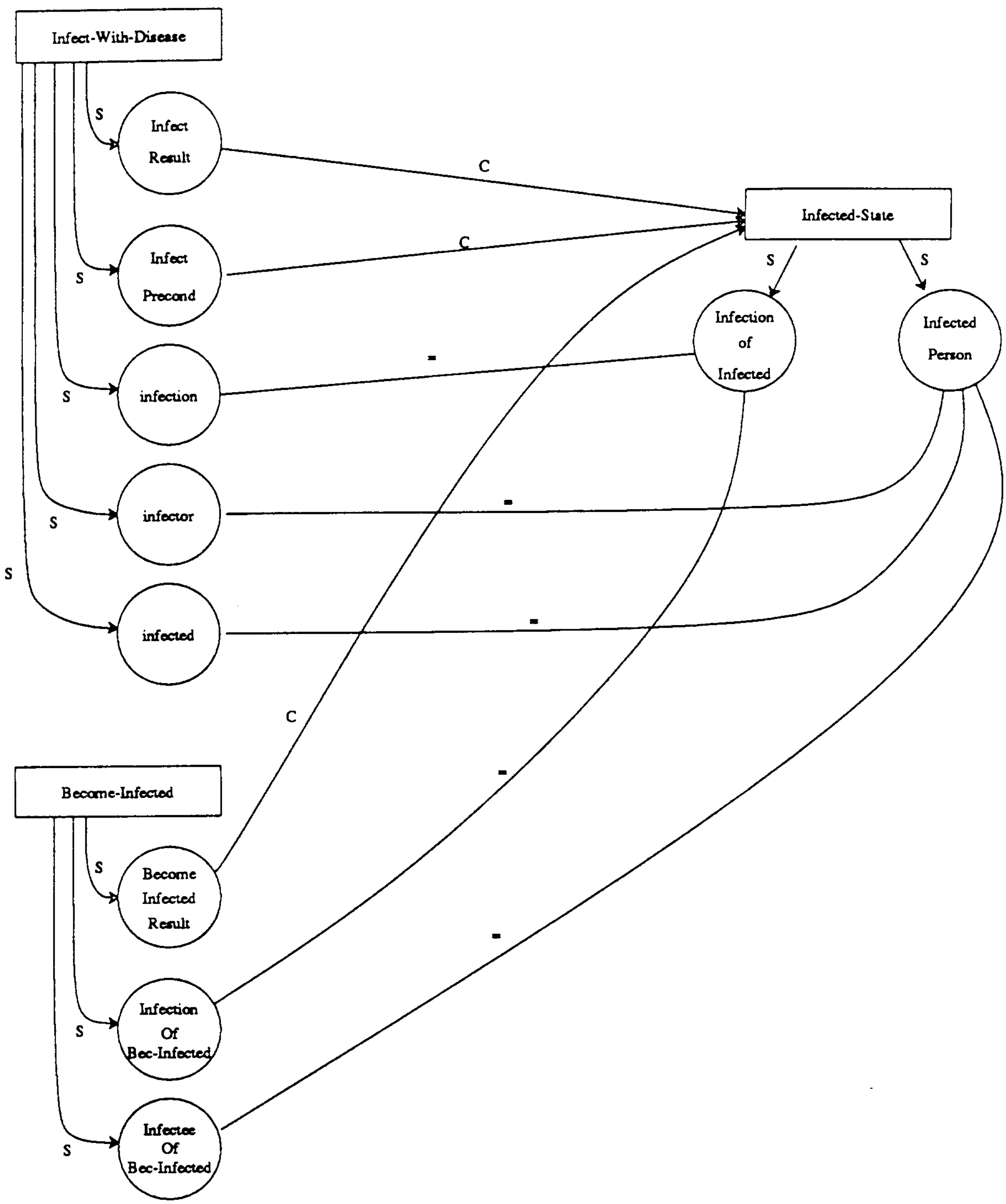

Figure 12: Infecting

The first condition to consider is the Relation condition. Consider the basis for the core-relation between `giving` and `having` in the source domain. This relationship is primarily based on the `giving-result` relation. This is a domain specific relation whose aspectuals are constrained to be a `giving` and a `having`, respectively. It is clear that this relation can not be preserved or carried over to the target domain intact. It is a fact about the domain of physical transfer and possession and has no coherent semantics in the domain of colds and infecting. However, some aspects of the relation are preserved in the target domain.

The relation in the target domain that the `giving-result` relation corresponds to is the `infect-action-result` relation. This follows directly from the `give-infection` metaphor-sense shown in Figure 3. The concept of `giving` is the source of the `giving-infecting` map whose target concept is the concept `infect-with-disease`. Correspondingly, the concept of `having` is the source of the `having-infected` map whose target concept is an `infected-state`. The `infect-with-disease` concept is directly linked to the `infected-state` via the relation underlying the `infect-result` slot. This is also a domain specific relation. Its aspectuals are constrained to be an `infect-action` and an `infected-state`.

The relations `giving-with-giving-result` and `infect-with-result` do share the common ancestor `result` relation. This is an abstract relation whose aspectuals are constrained to be an `event` and a `stative`. The conventional metaphorical use of core-related concepts will hierarchically reflect the relation from the source domain in the target domain. In this way, domain specific facts about the relations are ignored while a more abstract structural relationship is maintained.

In addition to hierarchically preserving the relation between two core-related concepts, the metaphor will also preserve the structured sharing imposed by the Containing condition. Remember that `giving` and `having` are core-related because they satisfy both the Relation Condition and the Containing Condition. The Containing Condition is satisfied because the concept of `giving` completely contains the concept of `having`. This was reflected in KODIAK by the fact that all the slots of `having` were equated to slots in the concept `giving`. This structure is preserved by the metaphorical use of *give* and *have* in (4). Figure 12 shows that all the slots of the `infected-state` concept, which corresponds metaphorically to `having`, are equated to slots of the `infect-with-disease` concept which corresponds to the concept `giving`.

4.5.3. Extended Metaphors

Consider the following examples.

(5) John gave Mary a cold.
(6) Mary has a cold.

The metaphor-sense accounting for (5) is shown in Figure 13. This metaphor-sense represents the conventional metaphor that to infect can be viewed as a transfer. The relevant maps to this discussion are the `givee-infected`, `given-cold`, and the `give-res-inf-result` mappings. These maps represent the ideas that the recipient of the giving is viewed as the newly infected person, the given thing is viewed as the cold, and the result of the giving is the result of the infecting action. Now consider the metaphor-sense underlying (6), as shown in Figure 14. In this case, the haver is the person with an infection, the cold is the thing had, and the having corresponds to the state of being infected.

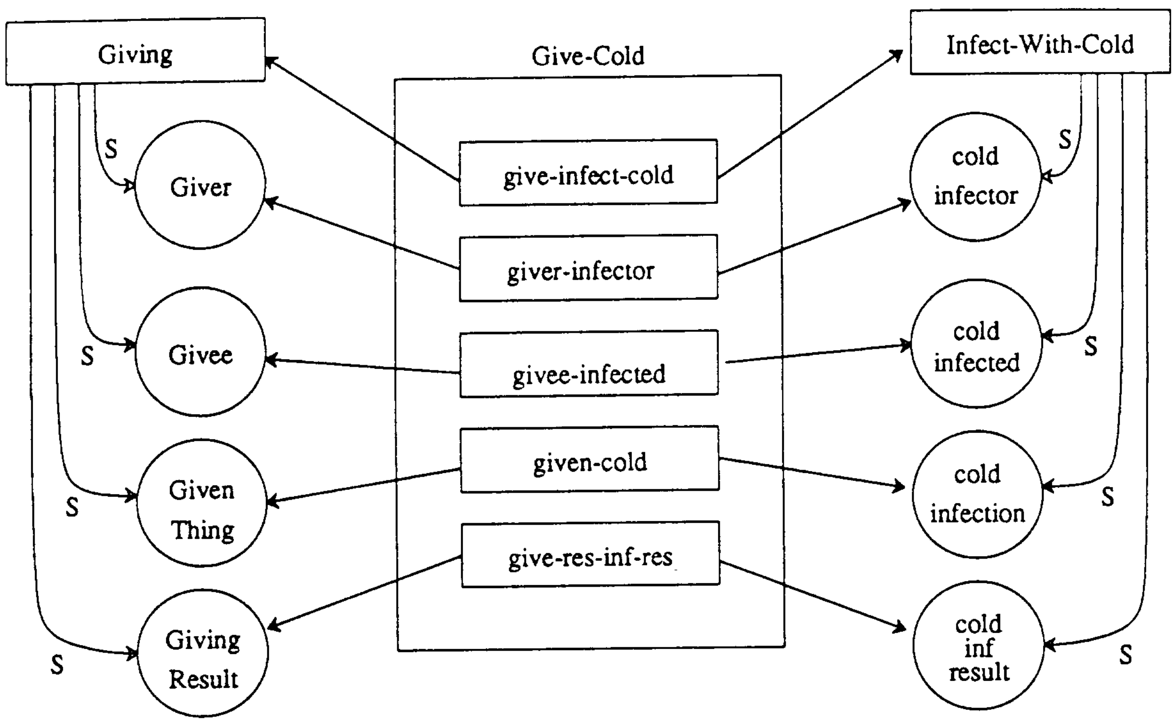

Figure 13: Give-A-Cold

This section will consider the details of how the `Have-Cold` and `Give-Cold` metaphors satisfy the following metaphorical extension requirements, introduced in Chapter 3.

Subset Requirement: The extended sense must contain all the core associations as a subset of its associations.

Closeness Requirement: The source and target concepts of the additional associations must be "closely connected" to the source and target concepts specified by the core associations.

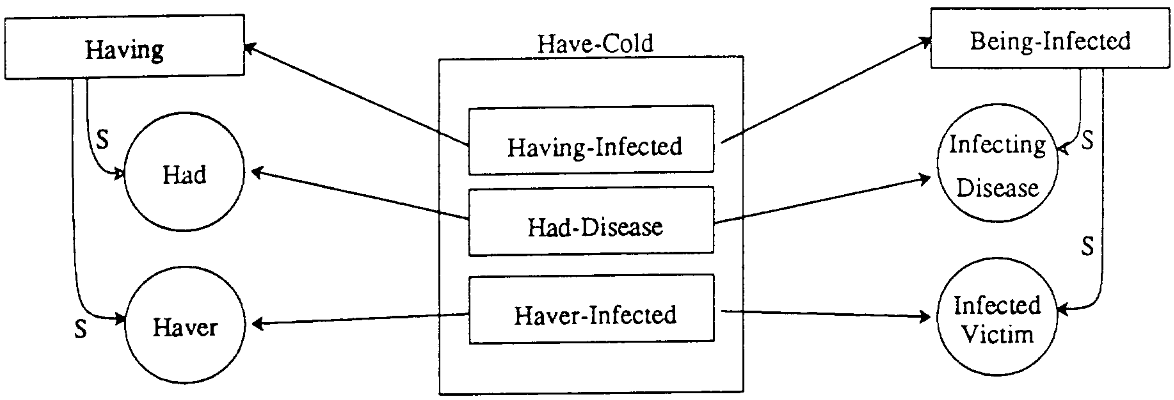

Figure 14: Have-A-Cold

Coherence Requirement: The new source and target concepts must be connected to the core source and targets in a coherent manner.

Figure 15 shows the extension relationship between the core metaphor `Have-Cold` and the extended sense `Give-Cold`. In this diagram, we see that the component metaphor maps from the core sense are shared and specialized by the extended sense. This example will be used to illustrate, in general, the mechanisms that are used to accomplish metaphor extension.

The metaphor-map representing the fact that a `haver` plays the role of the `infected`, the `haver-infected` map, appears twice in the extended sense, once as the `giver-infector` and again as the `givee-infected`. The `giver` of the `Giving` maps to the `Infector`; this matches the core map since the `giver` is equated to be a kind of `haver`. This core map is also used to represent the fact that the `givee` is also this kind of `haver` as a result of the `giving`. The next map to consider from the core is the `having-infected` map. This map also appears twice in the extended sense. The first use is in the `give-res-inf-res` map. This map indicates that the result of the `giving`, a `having` by the `givee`, plays the role of the result of the `infecting`, an infection of the infected person. It is, therefore, a specialized map from a `having` to an `infected-state`. It appears again in the `give-pre-inf-pre` metaphor map. This map indicates that you have to have a cold before you can give it to anyone. This is again basically a map from a `Having` to an `Infected-State`. The final map to consider is the core `had-cold` metaphor-map. This map appears once in the extended sense as the `given-cold` metaphor.

The links between the shared maps in the extended sense and the maps in the core sense are shown simply as individual D links. In fact, the specialization of a metaphor-map is a more complex structured association involving the specialization of each of the components. The following section on metaphor hierarchies will go into the details of

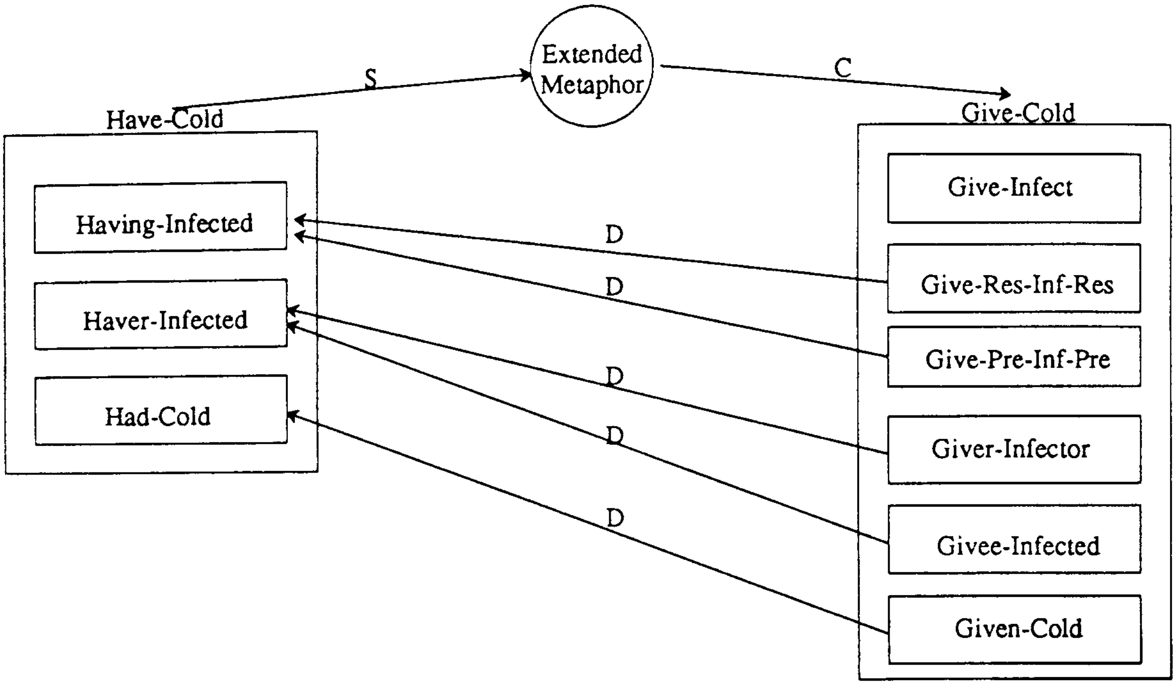

Figure 15: Having Extending to Giving

how this specialization is actually accomplished. At this point, it is sufficient to say that the first extension requirement is satisfied if all of the metaphor-maps in the core exist in the extension, as specializations.

The second requirement for extension is that the source and target components of the extended sense must be closely connected to the concepts specified by the core. Figure 11 shows that this is clearly fulfilled, in this case, in the source domain. The concepts in the source domain of the extended sense are all either equated to, immediately dominated by, or related by a single relation to the concepts in the source of the core domain. As shown in Figure 12, the same is true in the target domain. In general, the second requirement will be satisfied if the extended concepts are related to the core by these types of close association.

The final requirement specifies that the extended concepts be consistent extensions from both the source and target domain of the core metaphor. This is met, in this case, by the `result` relations in both the source and target. On the source side, the `Giving` is related to the `Having` from the core by a result relation. This is shown in the abbreviated slot form in Figure 11 by the `giving-result` slot. This relation is inherited from the more general category of `Action` and indicates that a result of a `Giving` is a `Having`. The same analysis holds for the target domain. The result of an `Infecting` is a state called `Being-Infected`. This last requirement is, therefore, satisfied if hierarchically related relationships hold between the core and extended concepts in both the source and target domains. In this case, it is a kind of `Result` relation.

The final relation to be discussed from Figure 15 is the `extends-to` relation shown with the slot notation. This relation is used as an explicit link from a core sense to an extension of that sense. In this case, it goes from the `Have-Cold` metaphor to the `Give-Cold` metaphor. This relation is in a sense redundant. If the extension requirements hold then it is possible to compute whether or not a given sense is an extension to another sense. The `extends-to` relation is nevertheless represented explicitly, in keeping with our overall representational philosophy of making as many relevant facts as possible explicit. This relation will be used directly by the learning system in determining the scope of possible new metaphors.

4.6. Metaphor Hierarchies

Metaphor-maps and metaphor-senses are all full-fledged KODIAK concepts, and can therefore be arranged in abstraction hierarchies. Hierarchies are the primary mechanism used to account for the similarities and differences among conventional metaphors. The following discussion will begin by showing the hierarchical relationships among various metaphor-maps. The following section will then go on to show how the various similarities and differences among conventional metaphors follow directly from the relations among metaphor-maps.

Consider again the `kill-process` map from (1). This is a mapping from a concept constrained to be a living thing to a target concept that is not a living thing. This is a manifestation of a more general metaphor that allows us to view non-living things in terms of living things for the purpose of explaining or understanding their behavior in terms of living things. Examples (7) through (10) from Lakoff and Johnson (1981) all contain specialized instances of this general metaphor.

(7) Inflation is eating up our savings.
(8) Those ideas died a long time ago.
(9) He is the father of modern quantum mechanics.
(10) Those ideas won't yield any fruit.

Example (7) is motivated by the metaphor that the reduction in savings caused by inflation can be viewed as inflation consuming the savings. Inflation is viewed as an animal that can consume things. Example (8) contains a metaphor dealing with the duration of ideas. When an idea is no longer held or believed it has died. At a more specific level we have an idea filling the role of the `dier` of a `death-event` as in Figure 4. This role has the similar constraint as the `kill-victim` of (1) that it be a `living-thing`. In a similar way as in (1), a set of interrelated metaphor-maps link the source domain of death to the target domain of ideas and their duration. In particular, there is a metaphor-map from the dier to a concept that is not a living-thing.

Example (9) contains the notion that the creation of an idea is a birth-event, and that the originator of the idea plays the role of the father in the birth event with the created idea playing role of the child. Once again, in this metaphor, there is a metaphor-map from a concept that is not a living thing (the created idea) to a role that must be one (the child being born). This metaphor-map, however, is more specific since the constraint is not only to be a living thing but to be human. Finally, example (10) contains the notion that an idea can produce new ideas. This is metaphorically structured as a plant producing new fruit. In this case, an idea is viewed as a specific kind of living thing, a plant.

What (1) and (8) through (10) all have in common is the idea that an abstract concept like a process or idea can be viewed as a living thing to explain some aspect of its nature. They differ in the particular kind of living-thing that is used and in the role that it plays. These similarities and differences result in specific metaphor-maps in each particular case. What is needed is a mechanism that can capture the commonalities and differences among these various metaphor-maps.

This mechanism is provided by the general inheritance mechanisms provided by KODIAK. Metaphor-maps that share properties are dominated by more abstract parent maps that capture the commonalities among the children. The source and target aspectuals of the parent map are constrained by concepts that are more abstract than, and dominate, the constrainers on the children's source and target aspectuals. Figure 16 illustrates this situation with the maps from examples from (1) and (7) through (10).

The top half of Figure 16 shows the hierarchical relationships among the maps underlying the above examples. They all converge on the abstract metaphor-map representing the idea of viewing a non-living-thing as a living-thing. The two metaphor-maps in the dotted box are expanded in the bottom half of the diagram to show the exact details of the inheritance links. In this expansion, we can see that the `idea-as-living-thing` metaphor-map dominates the `forgotten-idea-as-dier` map. In the parent map, the `idea-as-target` aspectual is constrained to be an `idea`. The `living-thing-as-source` aspectual is constrained to be a `living-thing`. In the `forgotten-idea-as-dier` map, we see that the inherited source aspectual is specialized by being constrained by the `dier` aspectual of the `death-event`. The inherited target aspectual is further specialized by being constrained by the `forgotten-idea` aspectual. The constraining aspectuals `forgotten-idea` and `dier` satisfy the appropriate inheritance restrictions since they in turn are constrained by `idea` and `living-thing`, respectively. It is typically the case that the source and target aspectuals of metaphor-maps at the bottom of the metaphor hierarchy are constrained by the aspectuals of some specific context. More abstract metaphors tend to have their aspectuals constrained by less specific absolutes.

Careful examination of Figure 16 reveals that the metaphor-maps are actually being specialized in different ways. Consider the `process-as-living-thing` specialization of the `non-living-thing-as-living-thing`. In this case, the metaphor-map is specialized on the basis of the target concept. The generic category non-living-thing is replaced by process. In the case of the `non-living-thing-as-plant` mapping, the specialization is done on the basis of the source concept. The metaphor-map hierarchy may,

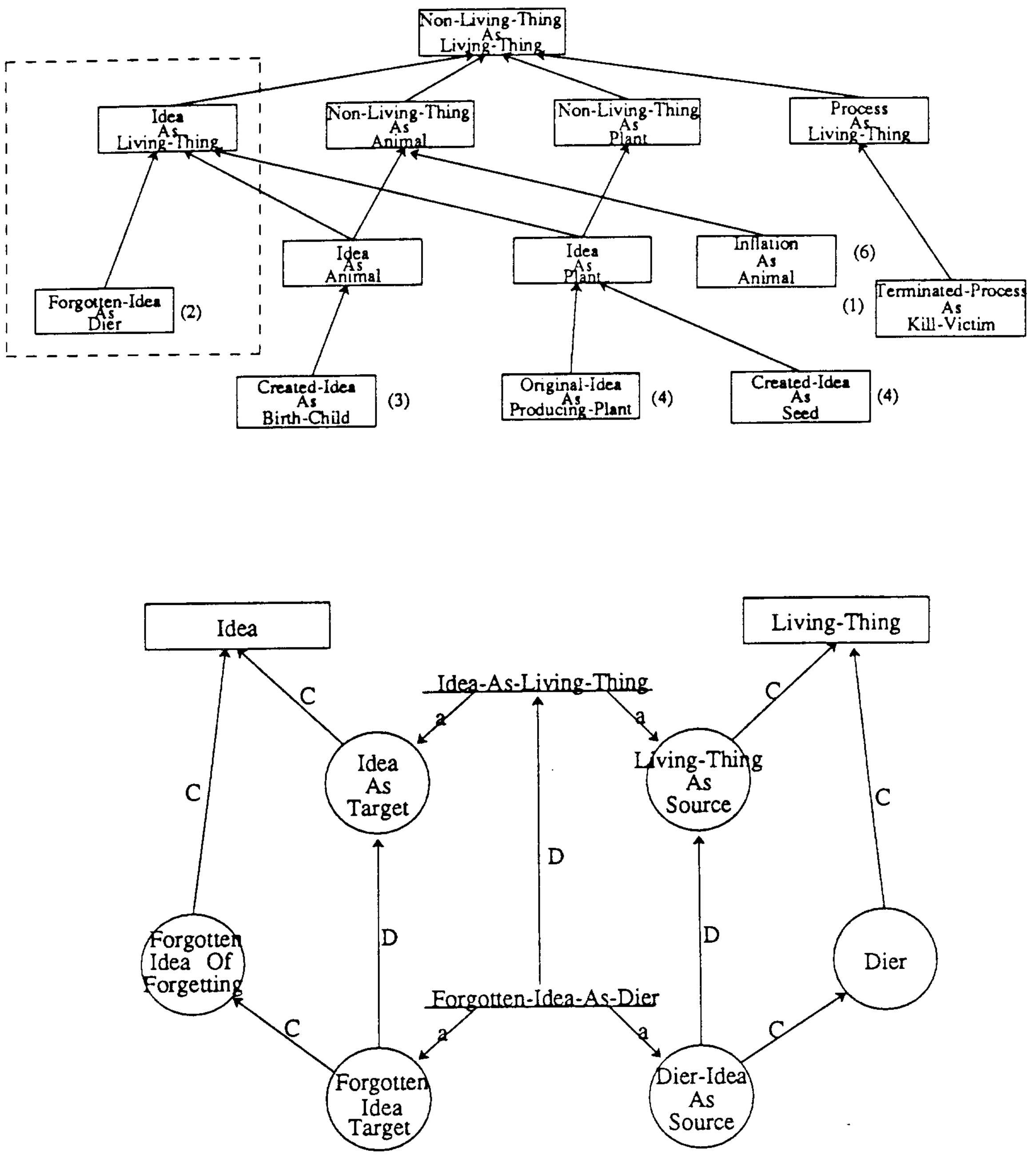

Figure 16: Metaphor-map Hierarchy

therefore, contain specializations based either on the source or target aspectuals or both.

Metaphor-maps may also participate in multiple inheritance. Consider the `idea-as-animal` and `idea-as-plant` maps. These maps inherit their respective specializations from multiple parents. The details for the `idea-as-animal` map are shown in

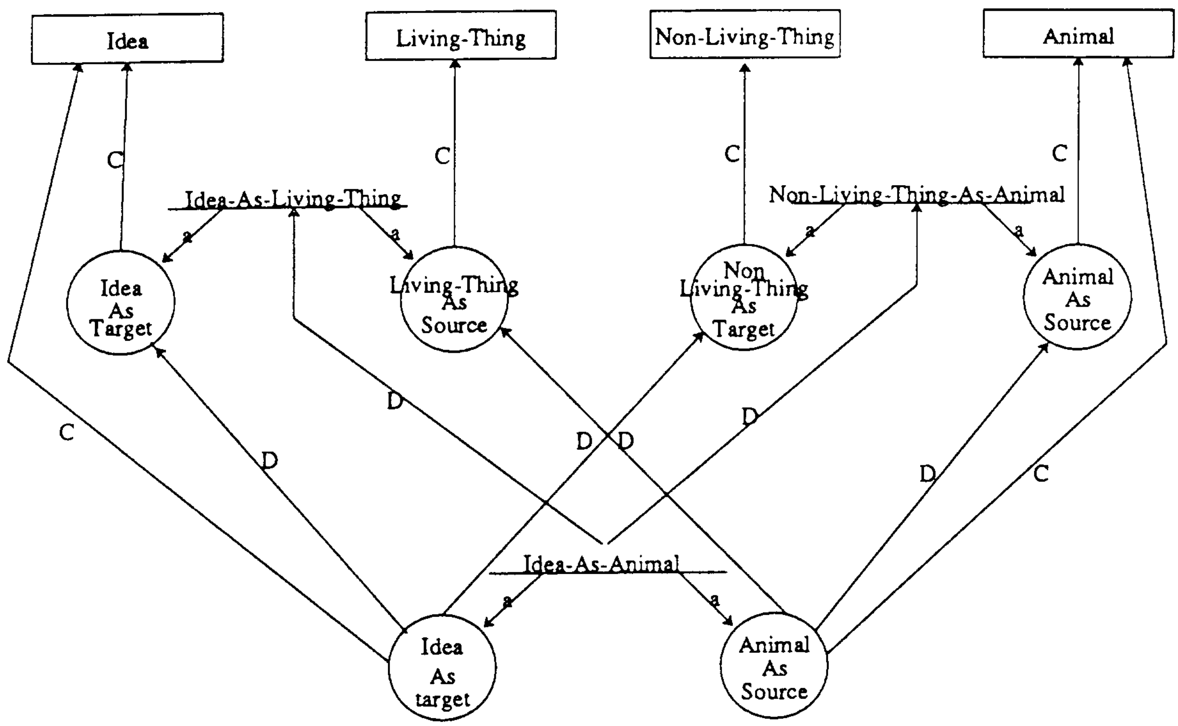

Figure 17: Multiple Inheritance

Figure 17. In this diagram, we can see that this map has two parents the `idea-as-living-thing` map and the `non-living-thing-as-animal` map. Its source and target aspectuals are each dominated by the both of the corresponding aspectuals from each parent. The multiple-inheritance mechanism assigns only the most specific constrainer to the lower child aspectuals. In this case, the source aspectual is constrained to be an `animal`, which is the more specific than `living-thing`. In the same way, the target is constrained to be an `idea` rather than the more abstract `non-living-thing`. In this way, the `idea-as-animal` map is created by specializing different parts of the two parent maps.

4.7. Metaphorical Interactions

At this point we have all the tools necessary to consider a wider range of interactions among conventional metaphors. This section will consider some interactions among different metaphors and how these interactions are reflected in the representation. The ideas presented in this section will serve to introduce some ideas that will be more fully discussed in Chapters 8 and 9. The emphasis here is in showing how the abstract connections among systems of conventional metaphors are reflected by the representation through the use of hierarchies.

Consider again the following examples discussed in Chapter 3.

(11) Mary gave John a good idea.
(12) Mary tried to sell the idea to John.

The metaphor-sense underlying the use of *give* in (11) consists of the following associations: communicating is transferring, the teller is the source and initiator of the transfer, the hearer is the receiver of the transfer, the communicated idea is the transferred object and that belief is possession of the object. The following associations underlie (12): persuading someone of some proposition is selling something to them, the goods sold is the proposition, the seller is the persuader, the customer is the person persuaded, and finally ownership is belief. Both of these metaphor-senses are instances of the abstract conduit metaphor as discussed by Reddy (1981). They share the same basic underlying structure that communication between a speaker and a listener can be viewed as a transfer between them. However, they each instantiate this basic structure with different, but related, source and target domains. What is needed is a way to account for this basic structure and the relationship between the senses of *give* and *sell* underlying (11) and (12).

The relationship between these senses can be clearly seen if we examine the two source and target domains. The source concepts giving and selling are both children of a more abstract concept denoting transfer where the donor plays the role of the actor. We can say that they are siblings with respect to the same parent category. They differ in the facts that are independently asserted about each. In a similar fashion, telling and persuading are also siblings, as children of a more abstract communication concept. Note that, as discussed in Chapter 3, both selling and persuading are more complex concepts then the corresponding concepts from (11). They inherit their complex definitions from other categories, in addition to the parents they share with the concepts from (11). Since both the sources and targets are siblings of common parents it seems to make sense to say that the metaphor-senses are also instantiations of a more abstract parent. This more abstract sense captures the basic commonality in the two metaphors. The requirement for such a common parent metaphor is that both the source and target concepts in each of the metaphors must be in a corresponding parent-child relation to a more abstract parent.

Figure 18 shows the hierarchical relationship among the senses underlying (11) and (12). It shows how each of the maps in the give and sell senses is properly dominated by a map in the more abstract `transfer-idea-metaphor`. Remember that a metaphor-sense is a KODIAK object consisting of a set of component-map relations. Each component-map relation connects a sense to a metaphor-map. Therefore, in order to accomplish the hierarchical organization desired for this example, each component-map of the sibling metaphor-senses must be dominated by a corresponding map in the parent sense. Correspondingly, each metaphor-map consists of a source and target aspectual. Each map may be dominated in the parent only if each of its aspectuals can be. Therefore, the well-formedness of the hierarchy is ultimately enforced at the level of the metaphor-map. If each of the senses can satisfy this requirement then the metaphor-senses themselves may be dominated by the more abstract sense.

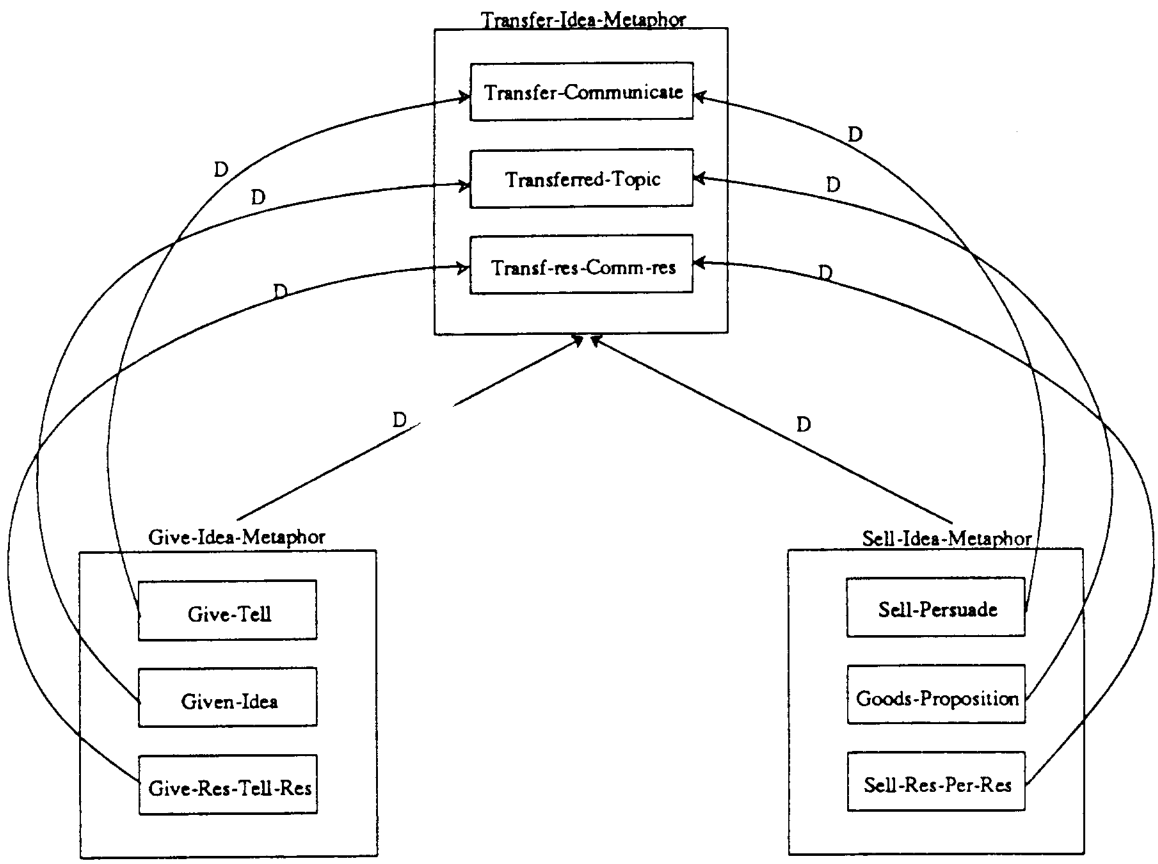

Figure 18: Giving and Selling Ideas

Finally, consider the relationship between the schemas underlying (13) and (14).

(13) Mary gave John a good idea.
(14) Mary gave John a cold.

These examples do not involve metaphors with overlapping scope, nor are they related in a strictly hierarchical fashion. Nevertheless, they do to have a structural similarity that needs to be represented.

In particular, note that both contain a map that represents the notion that something that is not an object can be treated as one, (a disease and an idea.) Both contain the notion that transfer represents some change of state with respect to the metaphorical object. The point of these examples is that the metaphor senses underlying Examples (13) and (14) contain individual component maps that are similar. This similarity is accounted for by the hierarchical relationship among the component maps in each schema. Figure 19 shows some of the relevant relationships between these schemas that account for their similarity. (Note that for the sake of readability most of the maps in each schema are omitted from this figure.)

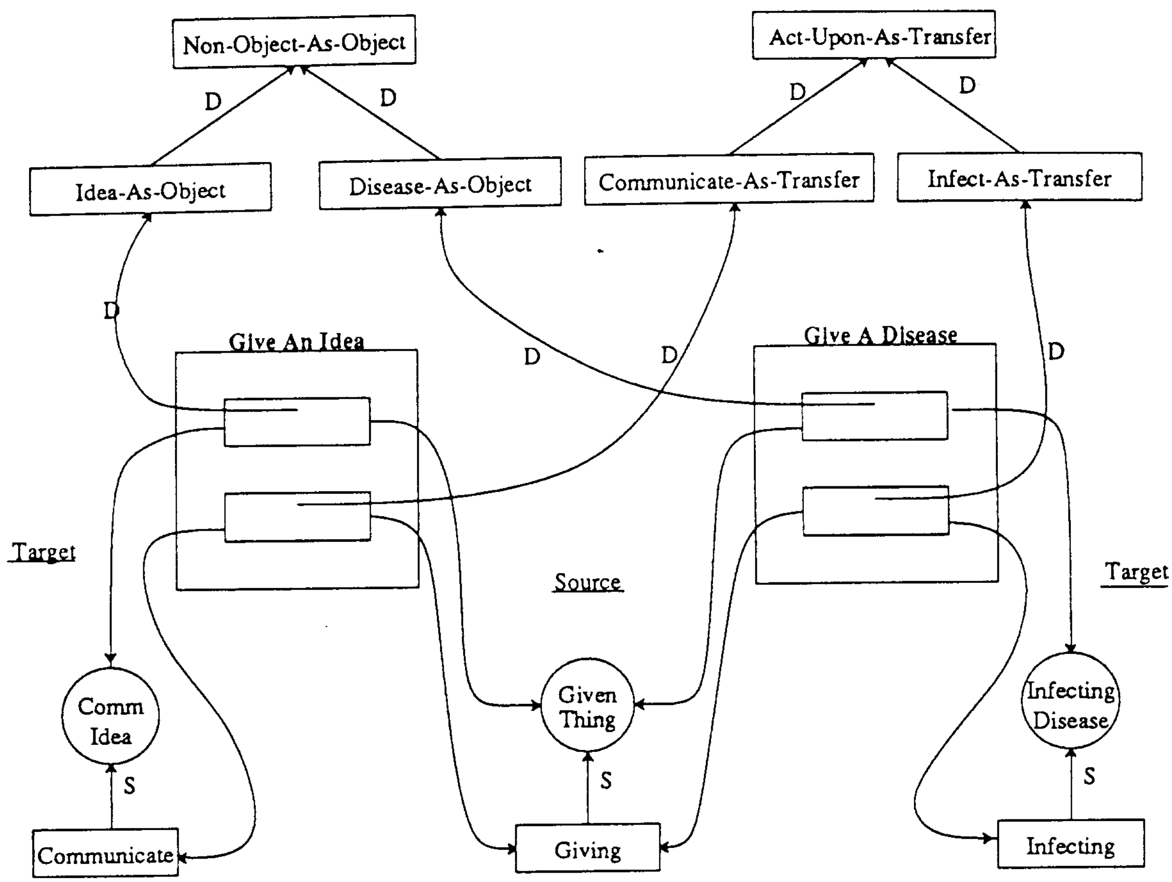

Figure 19: Giving Metaphors

This figure graphically illustrates the building block relationship between metaphor-maps and senses. In the `give-an-idea` sense, two maps are shown. The `give-tell` map is dominated by the `communicate-as-transfer` map which in turn is dominated by the `act-upon-as-transfer` (Jacobs 1985) map. The `act-upon-as-transfer` map is a map representing a general metaphor that allows abstract events with participants to be viewed as transfer-events where some participant receives or donates some metaphorical object. The `given-idea` map is dominated by the `idea-as-object` map which is in turn dominated by the `non-object-as-object` map. This metaphor map reflects the widespread use of reification in our conceptual system.

Similarly in the `give-a-disease` sense, two maps are shown. The `given-disease` map is dominated by the `disease-as-object` map. This in turn is a sibling of the `idea-as-object` map since they both have the `non-object-as-object` mapping as a parent. Finally, the `give-infect` map is dominated by the `infect-as-transfer` map which like the `communicate-as-transfer` map is dominated by the `act-upon-as-transfer mapping`.

In the discussion of examples (11) and (12) we saw the situation where two schemas were in a direct hierarchical relationship by virtue of the fact that all the component metaphor-maps of one schema were specialized in the more specific schema. Figure 19 illustrates the somewhat more distant relationship between the schemas underlying (13) and (14). They involve the same source domain with completely separate target domains. The domains of communication and infections are not related in any meaningful way in the hierarchy. They do not share any close meaningful parent categories. For this reason there can be no abstract metaphor schema dominating both of the schemas in question. However, the similarity of the two schemas is accounted for by the hierarchical structure of the component maps and the sharing of maps with common parents.

4.8. Summary

This chapter has introduced KODIAK and the mechanisms for representing metaphors. In particular *metaphor-maps* and *metaphor-senses*. In introducing these mechanisms I have touched upon some of the regularities among conventional metaphors that have motivated them. In particular, I have shown how the notion of abstraction hierarchies can be applied to the representation of conventional metaphors.

The following chapters will explore how the knowledge presented in this chapter can actually be used. Chapter 5 will show how an interpretation system can apply these structures to interpret metaphoric language. Chapters 6 through 10 show how these metaphors can be acquired.

Chapter 5

Metaphoric Interpretation

5.1. Introduction

This chapter demonstrates how the metaphoric knowledge described in the previous chapters can be applied to interpret conventional metaphoric language. The main thrust of the approach is that normal processing of metaphor proceeds through the application of specific knowledge about the metaphors in the language.

Metaphor is a normal and conventional part of language. The interpretation of utterances containing metaphors should reflect this fact in the way that the metaphors are processed. In particular, the interpretation of metaphor should not be viewed as an exception to normal processing. As discussed in Chapter 2, previous approaches have treated metaphors as anomalous inputs that are only dealt with when ordinary or literal interpretations are not coherent. The approach taken here is that metaphoric and literal interpretations have equal status and are evaluated using interpretation mechanisms that are fundamentally the same.

5.2. Conceptual Analysis

The interpretation of sentences containing metaphoric language is a two step process. In the first step, a syntactic parse-tree and a preliminary semantic representation are produced. In the second step, this preliminary representation is replaced by the most specific set of concepts that can coherently explain the input.

The following sections give overviews of the basic tasks in performed in each step. Section 5.3 gives the detailed interpretation algorithm, illustrated with examples intended to highlight important issues raised by this approach to metaphor interpretation.

5.2.1. Initial Parse

The first step in the interpretation of an input sentence is the production of a syntactic parse and a preliminary semantic representation known as a primal representation (Wilensky 1987). This first step is accomplished by a syntactic parser initially implemented by Peter Norvig. The parser is a unification-based bottom-up chart parser. It has been augmented to produce KODIAK concepts, in addition to a syntactic parse of the sentence.

This parser uses no semantic information to guide the parse and makes no effort to resolve syntactic ambiguities. It returns all possible parses with the primal semantics of each. This architecture was chosen mainly for pragmatic purposes. A more appropriate architecture would bring semantic information to bear during the parse in order to resolve these ambiguities.

The decision to perform metaphoric interpretation only after the initial parse is completed is another pragmatic decision. Metaphoric knowledge could also have been brought to bear during parsing. More generally, it should be possible to do metaphoric interpretation during stages of processing that are not directly linked to the interpretation of a single utterance. For practical purposes, this thesis limits the application of metaphoric knowledge to the analysis of single sentence utterances.

The primal representation produced by the parser represents concepts derivable from knowledge in the grammar and lexicon available to the parser. The primal representation represents a level of interpretation that is explicitly in need of further semantic processing. It should not be confused with what has traditionally been called a literal meaning. The primal representation should be simply considered as an intermediate stage in the interpretation process where only syntactic and lexical information has been utilized.

The primal representation, at this stage of processing, represents a portion of what Wilensky (1987) called the Primal Content of an utterance. The Primal Content represents the meaning of an utterance that is derivable from knowledge of the conventions of a language, independent of context. The primal representation produced at this stage of processing includes that portion of the Primal Content derivable from the conventions of the language embodied in the grammar and the lexicon. Further interpretation and elaboration by the Metaphor Interpretation System brings metaphorical conventions to bear to produce a level of representation that corresponds to a more complete Primal Content.

Consider the following examples.

(1) John gave Mary a gift.
(2) John gave Mary a cold.

The primal representations for these examples are given in Figure 1, in the linear KODIAK form introduced in Chapter 4.

```
(A Giving1 (↑ Giving)
   (agent1 (↑ agent) (A John1 (↑ John)))
   (patient1 (↑ patient) (A Mary1 (↑ Mary)))
   (object1 (↑ object) (A gift1 (↑ gift))))

(A Giving2 (↑ Giving)
   (agent2 (↑ agent) (A John2 (↑ John)))
   (patient2 (↑ patient) (A Mary2 (↑ Mary)))
   (object2 (↑ object) (A Cold1 (↑ Cold))))
```

Figure 1: Primal Representations.

The primal representation, in these examples, has simply assigned new instances of the absolutes mentioned in the sentence to the appropriate thematic roles. These input roles are left at the level of thematic roles and no constraint checking is done on their fillers. The main concept has been assigned to the concept directly specified by the word *give*. This, however, does not indicate a commitment that the final interpretation will involve a `Giving`. The `Giving` here merely represents the fact that the word *give* plays a central role in the input sentences.

The similarity between the primal representations of these two sentences reflects the similarity in the surface forms. The following interpretation process will take these primal forms and produce two very different interpretations.

5.2.2. Interpretation

The interpretation process consists of deciding which of the concepts that exist in the system's knowledge-base can most coherently accept the constraints imposed by the primal input. In particular, the literal meaning and all the known relevant conventional metaphorical meanings are considered as possible interpretations. The concept that best accounts for the input replaces the primal representation as the ultimate representation of the input.

Two basic inference processes are used, either separately or in tandem, to accomplish this interpretation task; these are *concretion* (Norvig 1983, Wilensky 1983, Wilensky et al 1986) and *metaphoric unviewing* (Wilensky 1987). Briefly, concretion is an inference that replaces an abstract concept by a more specific concept. Metaphoric unviewing replaces a given concept that plays the role of a source concept in a metaphor, with the corresponding target concept.

Possible candidate interpretations arise from two main sources. Non-metaphorical candidates include the direct literal interpretation of the primal representation and any more specific descendents of that concept. Candidate metaphorical interpretations include any metaphor-senses that are directly attached to, or inherited by, the non-metaphorical candidate concepts. Concretion and metaphoric unviewing inferences are applied to this set of candidate metaphors.

Constraint checking is the key operation underlying both concretion and metaphoric unviewing. The input concepts specified in the primal representation are compared against the constraints of the candidate interpretations. Checking the constraints on an interpretation consists of insuring that the specified input filler of each role is coherent with the semantic constraints on that role. For a filler to coherently fill a role it must either be an instance of the concept that constrains that role or an instance of a descendent of the constraining concept.

Constraint checking in a concretion inference consists of insuring that the fillers of the roles in the current concept can satisfy the constraints on all the more specific roles in the more specific category. Metaphoric interpretations are evaluated in a similar fashion. The filler of the source role must be coherent with the semantics specified for the corresponding target role in the particular metaphor being applied.

Consider Example (1) again. The interpretation process finds that the specified role fillers in the primal representation satisfy the constraints on the literal interpretation of *give* as the concept `Giving`. This initial concretion results in the creation of the `Giving` concept shown as `Giving1` in the top half of Figure 2.

The concretion inference should, however, find the most specific concept that can accommodate the input. In this case, we find that `Giving1` can be replaced by the more specific `Giving` concept, `Gift-Giving`. A concretion inference is, therefore, a recursive procedure proceeding down the hierarchy to the most specific category possible. In this example, the concept `Gift-Giving` is known to be a kind of `Giving` where the role of the `given` must be a gift of some kind. In the current example, this is known directly from the use of the word *gift*. The final concreted representation of this example is shown as `Gift-Giving1` in the bottom of Figure 2.

In the case of Example (2), the interpretation that best accounts for the input results from a metaphoric unviewing inference. The metaphor `Give-Disease` is used to produce the representation shown as `Infect-With-Disease1` in the top part in Figure 3. Figure 4 shows the `Give-Disease` metaphor with all of its metaphorical mappings. This diagram shows that the role of the `giver` corresponds to the `infector`, the `givee`

```
(A Giving1 (↑ Giving)
   (giver1 (↑ giver) (A John1 (↑ John)))
   (givee1 (↑ givee) (A Mary1 (↑ Mary)))
   (given1 (↑ given) (A gift1 (↑ gift))))

(A Gift-Giving1 (↑ Gift-Giving)
   (gift-giver1 (↑ gift-giver) (A John1 (↑ John)))
   (gift-givee1 (↑ gift-givee) (A Mary1 (↑ Mary)))
   (gift-given1 (↑ gift-given) (A gift1 (↑ gift))))
```

Figure 2: Final Concretion Step

corresponds to the `infected`, and the `given` corresponds to the `infection`. All the specified fillers of the source roles in the input can coherently fill the target roles. In particular, note that `Cold1`, the filler of the source role `given1`, satisfies the `Disease` constraint on the target role `infection`, corresponding to role `given1`.

```
(A Infect-With-Disease1 (↑ Infect-With-Disease)
   (infector1 (↑ infector)  (A John2 (↑ John)))
   (infected1 (↑ infected)   (A Mary2 (↑ Mary)))
   (infection1 (↑ infection)  (A Cold2 (↑ Cold))))

(A Cold-Infect1 (↑ Cold-Infect)
   (cold-infector1 (↑ cold-infector)  (A John2 (↑ John)))
   (cold-victim1 (↑ cold-victim)      (A Mary2 (↑ Mary)))
   (infected-cold1 (↑ infected-cold)  (A Cold2 (↑ Cold))))
```

Figure 3: Infection Concretion

Once it has been determined that the target roles of the metaphor can be coherently filled by the input fillers, a new instantiation of the target concept is produced to replace the primal representation. This newly created concept is now subject to possible further interpretation via a concretion inference. This reflects the fact that the metaphor applied in the unviewing inference may have its target concept represented at a level which is more abstract than the given example.

In this example, the system has knowledge of a concept that is more specific than

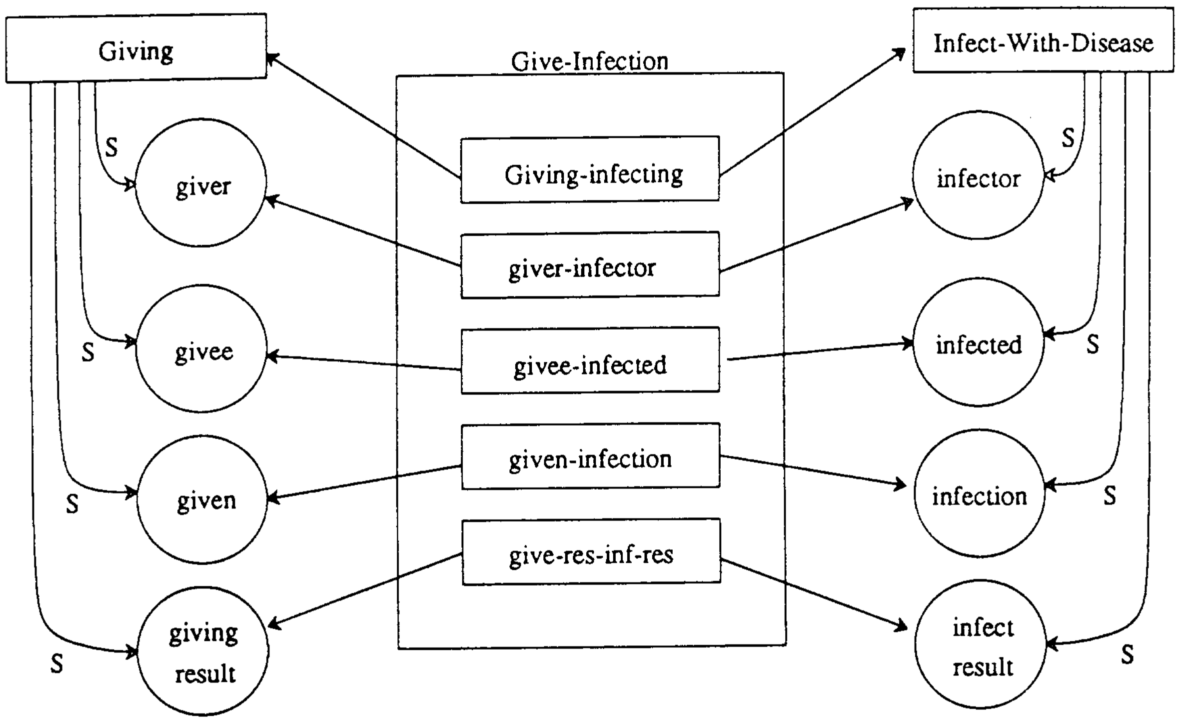

Figure 4: Giving a Disease

`Infect-With-Disease`. The concept `Cold-Infect` represents specific information about infecting someone with the common cold. The abstract target concept produced via the metaphoric unviewing inference is, therefore, concreted to this more specific concept. The final representation of the input is shown in Figure 3 as the concept `Cold-Infect1`.

5.3. Interpretation Algorithm

This section presents the basic interpretation algorithm. After each of the steps of the algorithm has been introduced, a series of detailed processing examples will be presented to illustrate the algorithm and to present various issues raised by the approach.

Step 1: Parser produces a primal representation for the input sentence.

Step 2: Case roles are concreted to the appropriate semantic relations associated with the primal concept.

Step 3: Collect all possible conventional interpretations, both metaphorical and literal, of the primal concept.

Step 4: Validate each of the possible interpretations. This consists of insuring that the concepts specified in the input satisfy the semantic constraints imposed by each of the possible interpretations.

Step 5: Apply all the consistent interpretations. This consists of the instantiation of the concepts underlying each of the possible coherent interpretations. This application may result in the replacement of the primal concept by either a specific literal interpretation or a conventional metaphorical one.

Step 6: Return all the interpretations that are consistent with the input concepts.

The most important point to realize about the strategy embodied in this algorithm is that the literal meaning of the input does not have a privileged status. Previous systems that have attempted to deal with metaphor have treated them as ill-formed exceptions to normal processing. As described in Chapter 2, these systems will only attempt a metaphorical interpretation when a violation of a selection restriction prevents a coherent reading for the literal meaning.

There are two main problems with this approach. The first is the fact that it gives an importance, or centrality, to the literal meaning over other conventional meanings that does not seem warranted. There seems to be no a priori reason for the interpreter to believe, or expect, the literal meaning over any other interpretation. The second problem is more immediate for these systems. There are conventional metaphors that do not exhibit surface selection restriction violations in their literal reading. What is needed, therefore, is a strategy that permits metaphoric interpretation that it is not sensitive to the coherence of a literal interpretation.

The strategy adopted here is to consider the literal meaning and all conventional metaphorical readings equally. The only requirement is that the given input must satisfy the requirements of the interpretation, whether literal or metaphorical. No privileged status is given to the literal meaning; it is checked along with all the other conventional meanings. Steps 3 and 4 of the algorithm collect and validate all the known conventional uses, metaphorical and literal. The recognition of the use of a conventional metaphor is in no way dependent upon the success or failure of the literal interpretation.

The steps of the algorithm and some of the issues raised by the strategy it embodies will now be illustrated in terms of the processing of the following examples.

(3) Inflation is *eating up* our savings.
(4) John *has* a cold.
(5) How can I *get into* lisp?
(6) McEnroe *killed* Connors.

For each of these examples, the system has direct knowledge of an appropriate conventional metaphor that will allow the sentence to be interpreted correctly. Consider the processing of Example (3), from Lakoff and Johnson (1980). In this metaphor, an abstract concept like *inflation* is viewed as an agent that can perform certain actions. In addition, the action of eating is viewed as an action or an event that results in the reduction in amount of the eaten thing.

```
> (do-sentence)
Interpreting sentence:

Inflation is eating up our savings.

Interpreting primal input.

(A Eating-Up61 (↑ Eating-Up)
   (agent595 (↑ agent)
   (A Inflation25 (↑ Inflation)))
   (patient560 (↑ patient)
    (A Savings25 (↑ Savings))))
```

In the first step, the parser accepts the input sentence as specified by the user and produces a primal representation of the input in the form of KODIAK concepts. The verb-particle construction, *eat up*, is treated as a single unit with a particular meaning.

```
Concreting input relations.

  Concreting  agent to eater-of-eating-up.
  Concreting  patient to eaten-up.
```

In the second step, based on information about the relations associated with the concept `Eating-Up`, the system determines that the relation `agent` found by the parser should be considered a `eater-of-eating-up`. In the same way, the patient is taken to be `eaten-up`. Note that these concreted relations still have the same preliminary status as the concept `Eating-Up`. These are all still primal concepts in need of interpretation. Concreting the relations at this point is merely a form of canonicalization to bring the case relations down to the same level as the main concept.

```
Interpreting concreted input.

(A Eating-Up61 (↑ Eating-Up)
   (eater-of-eating-up67
    (↑ eater-of-eating-up)
    (A Inflation25 (↑ Inflation)))
   (eaten-up39 (↑ eaten-up)
    (A Savings25 (↑ Savings))))
```

The case relations specified in the initial primal representation are now concreted to more specific relations that are directly attached to the concept `Giving`. This is called the *concreted primal representation*. Note that the constraints on the more specific relations are not checked against the fillers of the roles during this concretion.

In the third step of the algorithm, all the possible conventional interpretations of the input are collected for further consideration. In this case, the system finds two possible interpretations: the literal meaning and the `Eat-Up-Reduce-Money` metaphor.

```
Failed interpretation: Eating-Up61 as Eating-Up.

Valid known metaphorical interpretation.

Applying conventional metaphor Eat-Up-Reduce-Money.
```

Step 4 of the algorithm attempts to validate each of the possible candidate interpretations. In this case, it finds that the literal interpretation, represented by the concept `eating-up` is not consistent with the input. In particular, `Inflation` can not fill the role of eater because that role is constrained to be a `living-thing`. The concept `inflation` does not have `living-thing` as an ancestor in the hierarchy. In a similar fashion `savings` fails to satisfy the role of `eaten`.

The fact that the system considers the literal interpretation first is an artifact of the search procedure. It does not indicate any reliance on attempting the literal meaning first as was the case in previous approaches.

It does, however, find that there is a consistent metaphorical interpretation based on the `Eat-Up-Reduce-Money` metaphor. The same fillers of the input roles that could not satisfy the `Eating-Up` interpretation, can satisfy the corresponding target roles in this metaphor. In particular, `Inflation25` can fill the role of `money-reducer` and `Savings25` can fill the role of `money-reduced`.

```
(A Eat-Up-Reduce-Money  (↑ Eating-Metaphor Metaphor-Schema)
   (eaten-reduced eaten-up → money-reduced)
   (eater-reducer eater-of-eating-up → money-reducer)
   (eatup-reduce Eating-Up → Money-Loss))

 Mapping input concept Eating-Up61 to concept Money-Loss46
  Mapping input role eaten-up39 with filler Savings25 to
       target role money-reduced35
  Mapping input role eater-of-eating-up67 with filler Inflation25 to
       target role money-reducer57
```

In Step 5, each of the valid interpretations are applied to create new concepts to replace the primal concepts. The application of the `Eat-Up-Reduce-Money` metaphor results in the creation of a new instance of the main target concept Money-Loss with the input roles assigned according to the metaphor-maps `eater-reducer` and `eaten-reduced`.

```
Yielding interpretation:

(A Money-Loss46 (↑ Money-Loss)
   (money-reduced35 (↑ money-reduced)
    (A Savings25 (↑ Savings)))
   (money-reducer57 (↑ money-reducer)
    (A Inflation25 (↑ Inflation))))
```

The system next checks to see if the concept produced by the application of the metaphor can be concreted to any more specific concepts. In this case, there is the more specific concept `Savings-Loss` beneath `Money-Loss` that matches the constraints of the input example.

```
Concretion yields:

(A Savings-Loss20 (↑ Savings-Loss)
   (savings-reducer20 (↑ savings-reducer)
    (A Inflation25 (↑ Inflation)))
   (savings-reduced20 (↑ savings-reduced)
    (A Savings25 (↑ Savings))))
```

The concept `Eating-Up` has now been successfully interpreted as an instance of the concept `Savings-Loss`. No further interpretation is needed so this concept is returned as the final output.

```
Final interpretation of input:

(A Savings-Loss20 (↑ Savings-Loss)
   (savings-reducer20 (↑ savings-reducer)
    (A Inflation25 (↑ Inflation)))
   (savings-reduced20 (↑ savings-reduced)
    (A Savings25 (↑ Savings))))
```

In the previous example, the system found and applied an existing metaphor that was represented at a fairly specific level of detail. A relatively small concretion was made to concrete the target concept of the `Eat-Up-Reduce-Money` metaphor from a general loss of money to a `Savings-Loss`. The following example is an example of metaphorical interpretation combined with a more difficult concretion.

In this example, the system has knowledge of the highly abstract metaphor `Have-State`. This metaphor represents the widespread English metaphor that the concept of being in some state can be expressed as a possession. In this example, the state of being infected is expressed as a possession. The infected person is viewed as being in possession of the infection metaphorically. However, MIDAS does not have a specific metaphor representing this Infection-As-Possession use. Rather, it is forced to apply the more abstract metaphor `Have-State`. Concretion of this abstract concept leads to the intended concept of `Infected-State`.

```
> (do-sentence)

Interpreting sentence:

John has a cold.

Interpreting primal input.

(A Having7 (↑ Having)
   (agent52 (↑ agent) (A John49 (↑ John)))
   (patient52 (↑ patient) (A Cold24 (↑ Cold))))

Concreting input relations.

  Concreting  patient to had.
  Concreting  agent to haver.

Interpreting concreted input.

(A Having7 (↑ Having)
   (haver7 (↑ haver) (A John49 (↑ John)))
   (had7 (↑ had) (A Cold24 (↑ Cold))))


Failed interpretation: Having7 as Having.
Failed interpretation: Having7 as Have-Idea.
Failed interpretation: Having7 as Have-Permission.
```

The system, at this point, has determined that the literal interpretation and two metaphorical interpretations are not appropriate.

```
Valid known metaphorical interpretation.

Applying conventional metaphor Have-State.

(A Have-State (↑ Metaphor-Schema)
```

```
(have-state-map Having → State)
(had-state-value had → state-value)
(haver-state-holder haver → state-object))
```

The system finds that the abstract `Have-State` metaphor is applicable to the primal concepts.

```
Yielding interpretation:

(A State4 (↑ State)
   (state-value4 (↑ state-value)
    (A Cold24 (↑ Cold)))
   (state-object4 (↑ state-object)
    (A John49 (↑ John))))
```

The system applies the metaphor to yield an instance of the `State` concept. This concept merely indicates that the target concept is a state that holds between `John` and `Cold`.

```
Concretion yields:

(A Cold-Inf-State4 (↑ Cold-Inf-State)
   (cold-inf-person4 (↑ cold-inf-person)
    (A John49 (↑ John)))
   (cold-inf-of4 (↑ cold-inf-of)
    (A Cold24 (↑ Cold))))
```

The concretion process then replaces this abstract state concept with the most specific known kind of state that can accommodate the input values. In this case, the resulting concept is `Cold-Inf-State` representing the state of a person being infected with a common cold.

```
Final interpretation of input:

(A Cold-Inf-State4 (↑ Cold-Inf-State)
   (cold-inf-person4 (↑ cold-inf-person)
    (A John49 (↑ John)))
   (cold-inf-of4 (↑ cold-inf-of)
    (A Cold24 (↑ Cold))))
```

As mentioned earlier, in our strategy the recognition of the use of a conventional metaphor is in no way dependent upon the success or failure of the literal interpretation. Consider the following example.

(7) McEnroe killed Connors.

This is a straightforward use of the conventional metaphor that to kill someone in a competition means to defeat them. This metaphor is particularly evident in the sports pages of any newspaper. The obvious problem for a system using selection restriction violations is that this example does not involve any violation of the selection restriction on the literal meaning. The MIDAS strategy avoids this problem by attempting to find all conventional interpretations of the input. In the following example, the system discovers that there are two legitimate interpretations to this example.

```
> (do-sentence)
Interpreting sentence:

McEnroe killed Connors.

Interpreting primal input.

(A Killing144 (↑ Killing)
   (agent596 (↑ agent)
    (A Mcenroe46 (↑ Mcenroe)))
   (patient561 (↑ patient)
    (A Connors45 (↑ Connors))))

Concreting input relations.

  Concreting  patient to kill-victim.
  Concreting  agent to killer.

Interpreting concreted input.

(A Killing144 (↑ Killing)
   (killer89 (↑ killer)
    (A Mcenroe46 (↑ Mcenroe)))
   (kill-victim89 (↑ kill-victim)
    (A Connors45 (↑ Connors))))

Valid literal interpretation.
```

The literal interpretation of `killing144` as an instance of `killing` is found to be a valid reading. This follows from the fact that the input roles `McEnroe` and `Connors` completely satisfy the constraints on the `killer` and `kill-victim` roles.

```
(A Killing144 (↑ Killing)
   (killer89 (↑ killer)
    (A Mcenroe46 (↑ Mcenroe)))
   (kill-victim89 (↑ kill-victim)
    (A Connors45 (↑ Connors))))
```

At this point, the system has determined that the literal reading of this killing is consistent with the input. It, however, continues to check for other possibly coherent readings.

```
Valid known metaphorical interpretation.

Applying conventional metaphor Kill-Sports-Defeat.
```

The system finds that the known `Kill-Sports-Defeat` metaphor can adequately accommodate the input concepts. This metaphor is, therefore, applied with new target concepts instantiated and filled in to represent the new interpretation of the primal representation.

```
(A Kill-Sports-Defeat  (↑ Kill-Metaphor Metaphor-Schema)
   (killed-defeated kill-victim → defeated)
   (killer-defeator killer → defeator)
   (kill-defeat Killing → Sports-Defeat))

Mapping input concept Killing144 to concept Sports-Defeat60
  Mapping input role kill-victim89 with filler Connors45 to
        target role defeated60
  Mapping input role killer89 with filler Mcenroe46 to
        target role defeator62

Yielding interpretation:

(A Sports-Defeat60 (↑ Sports-Defeat)
   (defeated60 (↑ defeated)
    (A Connors45 (↑ Connors)))
   (defeator62 (↑ defeator)
    (A Mcenroe46 (↑ Mcenroe))))
```

MIDAS goes on to find that the remaining killing metaphors are not applicable. In each of the remaining cases, the filler of the `kill-victim` role fails to meet the requirements of the target role in the metaphor.

```
Failed interpretation: Killing144 as Kill-Conversation.
Failed interpretation: Killing144 as Kill-Delete-Line.

Choosing among ambiguous interpretations.

(A Killing144 (↑ Killing)
   (killer89 (↑ killer)
    (A Mcenroe46 (↑ Mcenroe)))
   (kill-victim89 (↑ kill-victim)
    (A Connors45 (↑ Connors))))

(A Sports-Defeat60 (↑ Sports-Defeat)
   (defeated60 (↑ defeated)
    (A Connors45 (↑ Connors)))
   (defeator62 (↑ defeator)
    (A Mcenroe46 (↑ Mcenroe))))
```

According to the options chosen by the user, MIDAS may either return all the ambiguous interpretations, or it may attempt to disambiguate the senses immediately. In this case, it attempts a disambiguation and returns the metaphorical interpretation. The disambiguation heuristics will be described below.

```
Choosing best answer.

Final interpretation of input:

(A Sports-Defeat60 (↑ Sports-Defeat)
   (defeated60 (↑ defeated)
    (A Connors45 (↑ Connors)))
   (defeator62 (↑ defeator)
    (A Mcenroe46 (↑ Mcenroe))))
```

The process of disambiguating this kind of example should properly be performed by an interpreter that has full access to the discourse context. A context-sensitive text inference system like FAUSTUS (Norvig 1986) could accept the ambiguous meanings from the interpreter and choose one based on how well it fit into the current context. There is really insufficient information in the single sentence given above to adequately choose one of the meanings. However, when the system is asked to choose among ambiguous interpretations it uses the following specificity heuristic.

> *Select the interpretation that most tightly matches the constraints posed by the input concepts.*

Consider the application of this heuristic in the processing of (7). In this example, the metaphorical interpretation most closely matches the input. Consider the constraints posed by the concept `Killing`, which is the basis for the literal reading. The concept `Killing` specifies that the `killer` is an animate and that the `kill-victim` is a living thing. The input concepts `McEnroe` and `Connors` easily satisfy these constraints. The metaphorical roles of the `Kill-Sports-Defeat` reading have the more specific constraints that the `killer` and `kill-victim` must be known to be competitors in some sport. The knowledge-base has `McEnroe` and `Connors` represented as such competitors. Although the input concepts match both interpretations, they match the metaphorical more tightly therefore that one is selected.

Whether or not the interpreter should be attempting to disambiguate these senses at this point is not really the issue. The important point is that making a metaphorical interpretation versus a literal one should be based on how well that interpretation fits the known information. It should not be dependent on whether or not the literal meaning is coherent. Contrast this with the approach taken in FAUSTUS (Norvig 1986). FAUSTUS only applied metaphoric knowledge in the presence of a constraint violation. As Norvig points out, this prevents the interpretation of this example as a `Sports-Defeat`.

The representation and use of conventional metaphoric knowledge is particularly important in abstract technical domains like operating systems. Such domains tend to contain many concepts that are structured metaphorically. The following example illustrates the integration of MIDAS with the UNIX Consultant system. The UNIX domain has proven to be a rich test-bed for testing the applicability of our approach to metaphor.

In the following example, a user has posed a question to UC using the conventional environment metaphor for interactive computer processes. MIDAS finds and applies the appropriate metaphor to produce an appropriate representation to give to UC. UC accepts the interpreted representation and answers the user's query. At the present time, MIDAS is attached to a small prototype version of UC built to demonstrate the applicability of the metaphor system.

```
> (do-sentence)
Interpreting sentence:

How can I get into lisp?

Interpreting primal input.

(A Entering50 (↑ Entering)
   (agent597 (↑ agent) (A I203 (↑ I)))
   (patient562 (↑ patient) (A Lisp58 (↑ Lisp))))
```

The input phrase *get into* is treated as a phrasal unit with a conventional meaning corresponding to `Entering`.

```
Concreting input relations.

  Concreting  patient to entered.
  Concreting  agent to enterer.

Interpreting concreted input.

(A Entering50 (↑ Entering)
   (enterer50 (↑ enterer) (A I203 (↑ I)))
   (entered50 (↑ entered) (A Lisp58 (↑ Lisp))))

Failed interpretation: Entering50 as Entering.
Failed interpretation: Entering50 as Enter-Association.
```

The literal interpretation and a known metaphor are rejected before the correct metaphor is found and applied.

```
Valid known metaphorical interpretation.

Applying conventional metaphor Enter-Lisp.

(A Enter-Lisp  (↑ Container-Metaphor Metaphor-Schema)
   (enter-lisp-res enter-res → lisp-invoke-result)
   (lisp-enterer enterer → lisp-invoker)
   (entered-lisp entered → lisp-invoked)
   (enter-lisp-map Entering → Invoke-Lisp))

Mapping input concept Entering50 to concept Invoke-Lisp30
  Mapping input role enterer50 with filler I203 to
       target role lisp-invoker30
  Mapping input role entered50 with filler Lisp58 to
       target role lisp-invoked30
```

The `Enter-Lisp` metaphor has been found and applied to the given input concepts. The main source concept is interpreted as an instance of the `Invoke-Lisp` concept according to the `enter-lisp-map`. The input roles `enterer` and `entered` are interpreted as the target concepts `lisp-invoker` and `lisp-invoked` respectively.

```
Yielding interpretation:

(A Invoke-Lisp30 (↑ Invoke-Lisp)
   (lisp-invoked30 (↑ lisp-invoked)
    (A Lisp58 (↑ Lisp)))
```

```
(lisp-invoker30 (↑ lisp-invoker)
 (A I203 (↑ I))))
```

This interpretation of the `Entering` concept is then used to fill the role of the `topic` role of the how question that constitutes the representation of the rest of the sentence.

```
Final interpretation of input:

(A How-Q207 (↑ How-Q)
   (topic206 (↑ topic)
    (A Invoke-Lisp30 (↑ Invoke-Lisp)
       (lisp-invoked30 (↑ lisp-invoked)
     (A Lisp58 (↑ Lisp)))
       (lisp-invoker30 (↑ lisp-invoker)
     (A I203 (↑ I))))))
```

This how-question, along with the reinterpreted topic concept, is then passed along to the next stage of UC processing. UC then prints the answer as follows.

```
Calling UC on input:

(A How-Q207 (↑ How-Q)
   (topic206 (↑ topic)
    (A Invoke-Lisp30 (↑ Invoke-Lisp)
       (lisp-invoked30 (↑ lisp-invoked)
     (A Lisp58 (↑ Lisp)))
       (lisp-invoker30 (↑ lisp-invoker)
     (A I203 (↑ I))))))

UC: You can enter lisp by typing lisp to the shell.
```

The instance of the concept `Invoke-Lisp30` is still represented in the knowledge-base as being an instance of a target concept of the `Enter-Lisp` concept. When UC's generator encounters such a situation, it generates an English description of the concept in terms of the corresponding source concept from the same metaphor. In this way, UC produces natural output that takes the users initial language into account. Therefore, UC chooses to describe the `Invoke-Lisp` concept in terms of the source concept `Entering`.

5.4. Summary

The kind of metaphor handled by MIDAS corresponds to a generalization of the class discussed by Gentner (1981, 1988). Gentner notes that it is typically the case that when there is a semantic mismatch between a verb and its arguments, the verb is being used metaphorically. She calls this the *Verb-Mutability Effect*. She shows that when children are presented with novel verb object pairings, the subjects usually reinterpret the verb metaphorically. In fact, it seems that this phenomenon properly corresponds to words that can be said to have valence. Consider the following examples.

(8) John is *in* Emacs
(9) This is a *big* idea.
(10) Einstein is the *father* of modern physics.

In (8), the preposition *in* is used metaphorically while its argument *Emacs* refers directly. Similarly in (9), the adjective *big* is being metaphorically applied to an *idea*. Example (10) illustrates the case where a noun like *father* that can be said to have valence is used metaphorically. Therefore, the actual surface-level metaphors handled by MIDAS correspond to a generalization of Gentner's class to include any part of speech that has valence.

The main thrust of this approach to metaphor interpretation is the application of specific knowledge about the conventional metaphors in the language. The initial parse of a sentence produces a primal representation that is essentially a set of constraints on the final representation derived from the grammar and lexicon. The main task of the interpreter is to find any interpretation of the input, metaphorical or literal, that is coherent with the constraints posed by this primal representation.

Chapter 6

Overview of Metaphor Extension

6.1. Introduction

The previous chapter detailed the process by which conventional metaphors are interpreted. The remaining chapters of this thesis are concerned with the acquisition of new metaphorical knowledge. When the interpretation algorithm described in Chapter 5 fails to find any interpretation coherent with the input, this is taken to be evidence that an unknown metaphor is being used.

The purpose of this chapter is to give an overview of the general approach to learning new metaphors and to the component of MIDAS that embodies this approach - the Metaphor Extension System (MES). The full details of this approach and the MES will be spread out over the next three chapters.

6.2. The Learning Approach

The approach taken here to the understanding of new or unknown metaphors is called the Metaphor Extension Approach. The basic thrust of this approach is that a new metaphor can best be understood by extending an existing metaphor in a systematic fashion to cover a new use. The basis for this approach is the belief that the known set of conventional metaphors constitutes the best source of information to use in understanding new metaphors. Underlying this strategy is the assumption that the new metaphor will be systematic with the existing metaphors in one of the ways described in Chapters 3 and 4.

Note that a metaphor that MIDAS cannot interpret may be an indication of one of two situations. It may signal the existence of a metaphor that is a conventional part of the language that the system has not yet acquired. Alternatively, it may reflect the use of a truly novel metaphor, a metaphor that would not be considered as a conventional part of the language. The metaphor extension approach makes no attempt to differentiate these situations. In either case, the ease with which the metaphor is understood will be a function of how close the new metaphor is to one of the known metaphors. If the new metaphor is truly novel and not based on existing metaphors then it will probably not be understood. By the same token, if it is a conventional and the system has no knowledge of other metaphors close to this new one, it will also fail.

The success of the approach, therefore, lies in the robustness and systematicity of the knowledge-base of known metaphors. As more and more systematic metaphors are added to the knowledge-base, the likelyhood of an unknown conventional metaphor being understood increases. Although novel poetic metaphors are beyond the scope of this thesis, the work of Lakoff (1986) and Turner and Lakoff (1988) shows that novel poetic metaphors are principally based on extensions to, and combinations of, metaphors that are already part of the language. This is further indication that detailed knowledge of the metaphors in the language is crucial to the understanding and acquisition of both conventional and novel metaphors.

6.3. Context

The Metaphor Extension Approach deliberately avoids the use of one the main sources of knowledge used in previous approaches, namely, the unrestricted use of contextual information. As described in Chapter 2, previous approaches to metaphor and to language acquisition have frequently assumed that the meaning of an unknown word or phrase is already a part of the discourse context. It has either been inferred from extra-linguistic information or from other pieces of text. The learning task was seen as assigning the meaning of the new lexical item to one of the small number of concepts already in the context.

There are a number of practical and theoretical reasons for the decision to eschew the use of contextual information. The main practical reason was that the UNIX Consultant testbed for the theory had no context wider than the immediate sentence in which the metaphor occurred. A more compelling reason has to do with the claims being made about the learning behavior itself. If it can be shown that certain kinds of learning are possible using only long-term semantic knowledge about existing metaphors and the contents of the sentence containing the new metaphor then that is a significant claim.

6.4. The Extension Approach

The systematicities described in Chapters 3 and 4 form the basis for the Metaphor Extension Approach. When a new use is encountered the system attempts to find a known metaphor that can be related to this new use by one of the known systematicities.

Three kinds of related metaphors are recognized, yielding three kinds of extension inferences: *similarity extension*, *core extension* and *combined extension*.

Similarity Extension:

Understanding a new metaphor by using a known metaphor that is judged to be similar to the new use.

Example:

Understanding *Kill a process* by using *Kill a conversation.*

Core Extension:

Understanding a new metaphor by using a known metaphor that is core related to the new use.

Example:

Understanding *Get out of lisp* by using *Get into lisp.*

Combined Extension:

Understanding a new metaphor by using a known metaphor that is both core-related and similar to the new use.

Example:

Understanding *Exit emacs* by using *Enter mail.*

Figure 1: Extension Inferences

6.4.1. Similarity Learning

Consider the situation where (1) is encountered and the interpretation system can find no appropriate metaphor to apply to account for this case. Assume, however, that the system does have knowledge of the specific `In-Emacs` metaphor, shown in Figure 2, that allows it to understand Example (2).

(1) I am *in* Lisp.
(2) I am *in* Emacs.

Using the `In-Emacs` metaphor to understand Example (1) is an instance of a *similarity extension* inference. This kind of inference extends a metaphorical structuring of a concept similar to the target concept of the current example. In this case, the idea is to use the known metaphor underlying the phrase *in emacs* to infer the meaning of *in lisp*. This type of inference is based on the conceptual similarity between the target concept of the new metaphor and the target concept of a known metaphor. The fundamental assumption is that *a metaphor with the same source concept being applied to similar target concepts will have a similar meaning*.

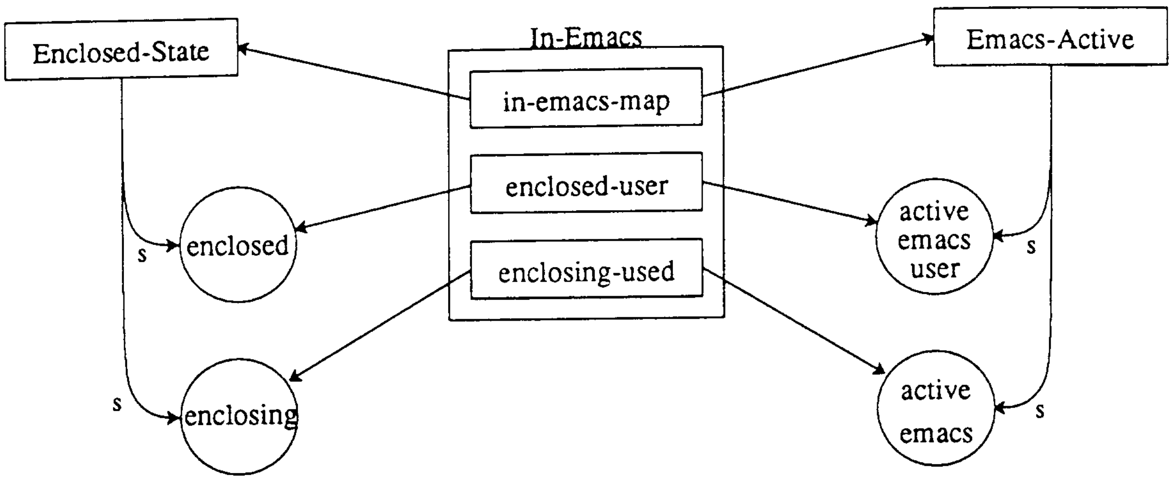

Figure 2: In-Emacs Metaphor

A similarity extension inference is a two step process of abstraction followed by concretion. The target concept of the candidate metaphor is abstracted to a concept that can accept the input target concepts of the new example. This abstract concept is then replaced by a more specific concept via a concretion inference. Remember that the concretion mechanism takes as input, a set of constraints on the concept being classified, and an abstract known ancestor of this concept. It returns the most specific concept beneath this initial ancestor that can accommodate the given constraints. The combined abstraction-concretion inference is similar to concretion in that it takes a set of constraints on the input concept and returns the most specific concept that can account for the inputs. It differs from concretion in that instead of being given an abstract ancestor of the final concept to search from, it is instead given an overly specific sibling or cousin from which to begin its search.

In this example, the target concept of the known metaphor, `Using-Emacs`, is replaced by the more abstract concept `Comp-Process-Active` during the initial abstraction step of the inference. This abstraction inference is intended to find the first ancestor of `Emacs-Active` that can accept the concept `Lisp` in the role played by the concept `Emacs`.

In the concretion step, the children of the `Comp-Process-Active` concept are examined to see if any of those involve the target concept of the new example, `Lisp`, in

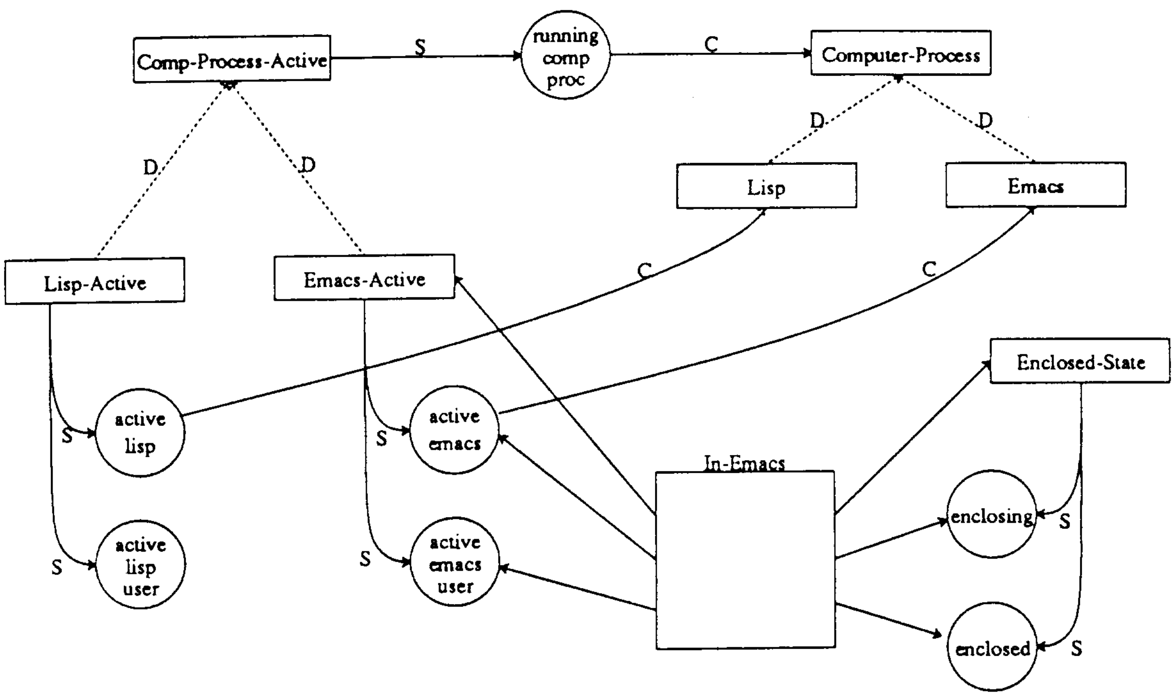

Figure 3: Emacs-Lisp Similarity Connection

an appropriate role. In the current example, the concept `Comp-Process-Active` is replaced by the more specific concept `Lisp-Active`, which represents the state of using the lisp system. Figure 3 shows the relevant pathway through the network that led to this concept.

The following annotated trace shows the actual processing of Example (1). In this example, the system learns the meaning of *in lisp* by applying a similarity inference to the known `In-Emacs` metaphor. The sections of the processing relevant to similarity inferences are annotated. The details of the overall algorithm are discussed below in Section 6.5.

```
> (do-sentence)

Interpreting sentence:

I am in lisp.

Interpreting primal input.

(A Enclosed-State3 (↑ Enclosed-State)
```

```
   (agent92 (↑ agent) (A I21 (↑ I)))
   (patient92 (↑ patient) (A Lisp4 (↑ Lisp))))

Concreting input relations.

  Concreting  patient to enclosing.
  Concreting  agent to enclosed.

Interpreting concreted input.

(A Enclosed-State3 (↑ Enclosed-State)
   (enclosed3 (↑ enclosed) (A I21 (↑ I)))
   (enclosing3 (↑ enclosing)
    (A Lisp4 (↑ Lisp))))


Failed interpretation: Enclosed-State3 as Enclosed-State.
Failed interpretation: Enclosed-State3 as In-Emacs.

No valid interpretations. Attempting to extend existing metaphor.

==========================================================
Entering Metaphor Extension System
==========================================================


Searching for related known metaphors.

Metaphors found: In-Emacs Entering-Invoke-Emacs Enter-Association

Candidate metaphor In-Emacs has ranking 2
Candidate metaphor Entering-Invoke-Emacs has ranking 3
Candidate metaphor Enter-Association has ranking 7

Selecting metaphor In-Emacs to extend from.

(A In-Emacs (↑ Metaphor-Schema)
   (in-emacs-map Enclosed-State → Emacs-Active)
   (enclosing-used enclosing → active-emacs)
   (enclosed-user enclosed → active-emacs-user))
```

At this point in the processing, the system has collected a relevant set of metaphors, and has chosen the `In-Emacs` metaphor for further processing.

```
Attempting a similarity extension inference.

Extending similar metaphor In-Emacs with target concept Emacs-Active.

Abstracting Emacs-Active to ancestor concept Comp-Process-Active producing
abstract target meaning:

(A Comp-Process-Active2
   (↑ Comp-Process-Active)
   (running-comp-proc2 (↑ running-comp-proc)
    (A Lisp4 (↑ Lisp)))
   (running-comp-proc-user2
    (↑ running-comp-proc-user) (A I21 (↑ I))))
```

The first step in the similarity extension inference is to abstract the target concept of the related metaphor to a point in the hierarchy where the current target concepts can be accommodated. In this case, the target concept `Emacs-Active` is abstracted to the `Comp-Process-Active` concept, representing the abstract notion of a process being used. This abstraction is primarily based on the target role `active-emacs` in the original target concept. This role is abstracted to the concept `running-comp-proc` in the concept `Comp-Process-Active`. The `running-comp-proc` role is constrained to be filled by a concept dominated by `Computer-Process`. This constraint can accommodate the specified target concept `Lisp`. The abstraction process stops at this concept.

```
Concreting target concept Comp-Process-Active to Lisp-Active
producing concreted meaning:

(A Lisp-Active2 (↑ Lisp-Active)
   (active-lisp2 (↑ active-lisp)
    (A Lisp4 (↑ Lisp)))
   (active-lisp-user2 (↑ active-lisp-user)
    (A I21 (↑ I))))
```

The abstract `Comp-Process-Active` concept is now concreted to find the most specific descendent that can take the input concepts. In this case, the most specific descendent concept of `Comp-Process-Active` that can take the input concepts is the `Lisp-Active` concept. A new instance of this concept is, therefore, created to represent the intended target meaning of the new use.

```
Creating new metaphor:

 Mapping main source concept Enclosed-State to main target concept
Lisp-Active.
  Mapping source role enclosed to target role active-lisp-user.
  Mapping source role enclosing to target role active-lisp.

(A Enclosed-State-Lisp-Active (↑ In-Metaphor)
   (enclosing-active-lisp-map enclosing → active-lisp)
   (enclosed-active-lisp-user-map enclosed → active-lisp-user)
   (enclosed-state-lisp-active-map Enclosed-State → Lisp-Active))


Final interpretation of input:

(A Lisp-Active2 (↑ Lisp-Active)
   (active-lisp2 (↑ active-lisp)
    (A Lisp4 (↑ Lisp)))
   (active-lisp-user2 (↑ active-lisp-user)
    (A I21 (↑ I))))
```

A full analysis of the nature and use of similarity among metaphors for the purposes of learning are given in Chapter 8.

6.4.2. Core Extensions

Consider the situation where (3) is encountered, and MIDAS can find no appropriate metaphor that can account for this use of *get into*. It does, however, have knowledge of the `In-Emacs` metaphor that allows it to understand (4).

(3) How can I get into emacs?
(4) I am in emacs.

Using the `In-Emacs` metaphor to understand Example (3) is an instance of a *core-extension* inference. A core-extension inference extends an existing partial metaphorical structuring of a target concept when a new metaphor is encountered that is core-related to the already known metaphor. Briefly, concepts may be said to be core-related when one of the concepts is, in effect, completely contained within the definition of the other. In the case of Examples (3) and (4), the concept `Entering` completely contains as a subpart the concept of `Enclosing`, underlying (4). Two concepts may also be said to be core-related if they share a common contained concept.

A core-extension inference exploits the core-relationship between the source concept of the known metaphor and the source of the new metaphor to identify the intended target concept of the new example. The source core-relationship is used by matching it against relations directly attached to the target concept of the known metaphor. The relation in the target domain that best matches the source relation hierarchically is taken to point at the intended target concept.

The basis for this approach is that conceptual core-relationships among words in the source domain will be at least partially preserved in the target domain. Remember that this preservation of core-relations is captured by Metaphor Preservation Principle, introduced in Chapter 3.

> *Metaphor Preservation Principle:* The metaphorical use of core-related words in a given target domain will hierarchically preserve the core-relationships established among the corresponding words in the source domain.

The matching part of a core extension inference is essentially an analogical matching inference based upon an abstraction hierarchy. The fundamental assumption is that there will be a common structuring of the concepts in the source and target domains at some level of abstraction. Note, however, that this analogical matching is done in the context of an existing partial metaphorical structure. The known metaphor establishes a highly constrained object correspondence, as a context for further matching. It is this initial partial structure provided by the previously understood metaphor that makes the matching possible.

In the current example, the system is not applying a blind analogical match from the domain of enclosures to the domain of computer processes. Rather the analogical match is of the form:

In:Enter::Using-Emacs:?

In other words, given the fact that we know what *in emacs* means, we must determine which target concept best explains the meaning of *enter emacs*. It is the task of the core-extension inference is to find this target concept.

In this case, the `result` relationship between the source concepts `Entering` and `Enclosed-State` is used to find the intended target concept `Invoke-Emacs`. The `result` relation between the concepts `Entering` and `Enclosed-State` is matched against the relations attached to the target concept `Emacs-Active`, yielding the concept `Invoke-Emacs`. Figure 4 shows the relevant relationships in this example.

The following annotated trace shows the processing of Example (3). In this example, the system learns the meaning of *get into emacs* by applying a core-extension inference to the known `In-Emacs` metaphor. The sections of the processing relevant to core-

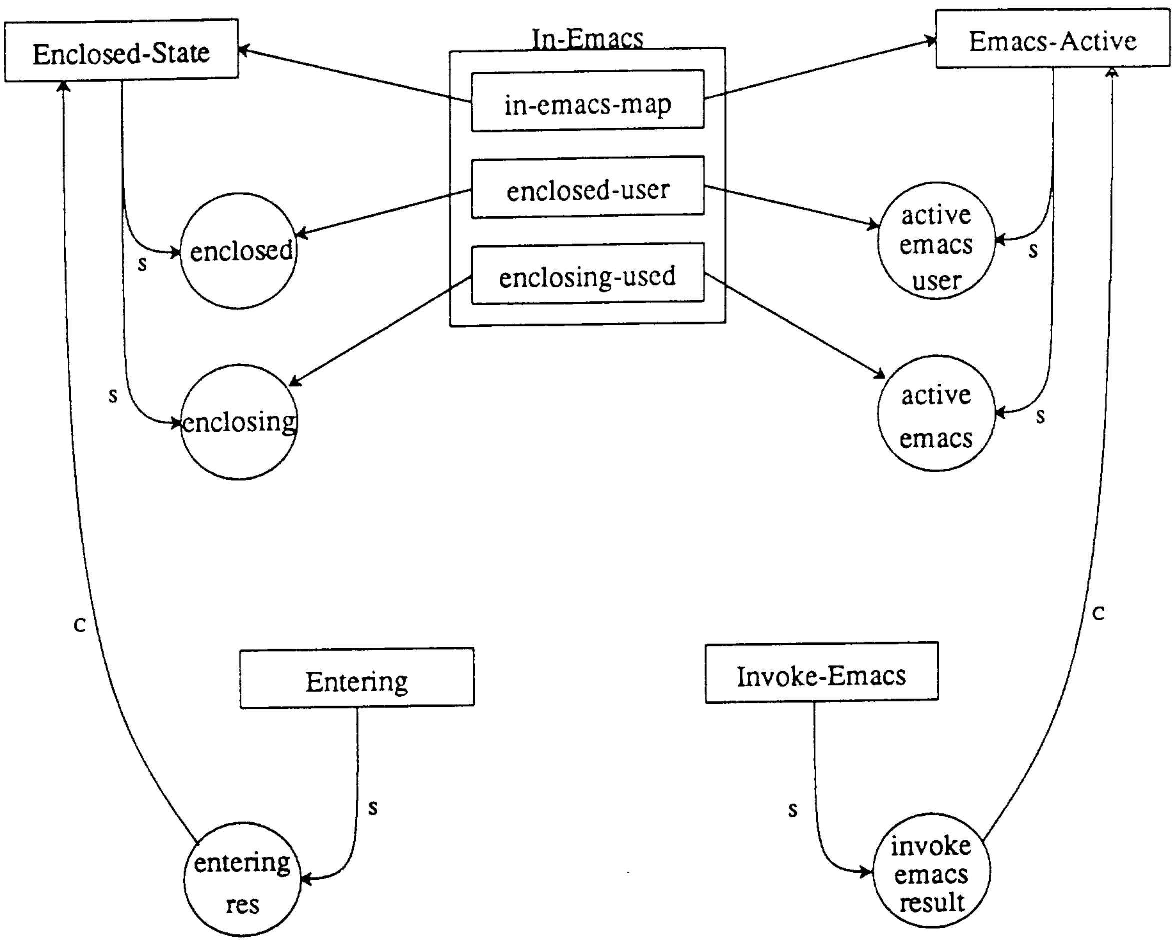

Figure 4: Entering Enclosed Core Connection

extension inference are annotated. The details of the overall algorithm are discussed below in Section 6.5.

```
> (do-sentence)

Interpreting sentence:

How can I get into emacs?

Interpreting primal input.

(A Entering4 (↑ Entering)
   (agent91 (↑ agent) (A I20 (↑ I)))
   (patient91 (↑ patient)
```

```
    (A Emacs12 (↑ Emacs))))

Concreting input relations.

  Concreting  patient to entered.
  Concreting  agent to enterer.

Interpreting concreted input.

(A Entering4 (↑ Entering)
   (enterer4 (↑ enterer) (A I20 (↑ I)))
   (entered4 (↑ entered) (A Emacs12 (↑ Emacs))))


Failed interpretation: Entering4 as Entering.
Failed interpretation: Entering4 as Enter-Association.

No valid interpretations. Attempting to extend existing metaphor.

===========================================================
Entering Metaphor Extension System
===========================================================


Searching for related known metaphors.

Metaphors found: In-Emacs Enter-Association

Candidate metaphor Enter-Association has ranking 6
Candidate metaphor In-Emacs has ranking 2
```

The system finds and selects the existing `In-Emacs` metaphor to use to learn this new use.

```
Selecting metaphor In-Emacs to extend from.

(A In-Emacs (↑ Metaphor-Schema)
   (in-emacs-map Enclosed-State → Emacs-Active)
   (enclosing-used enclosing → active-emacs)
   (enclosed-user enclosed → active-emacs-user))

Attempting a core-related metaphorical extension.

Extending similar core-related metaphor In-Emacs
with target concept Emacs-Active.
```

```
This is a direct extension inference.
```

The core-extension inference first finds the core connection between the source concept of the new metaphor and the source of the known metaphor. This connection is then applied to the target concept of the `In-Emacs` metaphor to find the appropriate filler of the path among the target concepts.

```
Applying source path:

Entering → enter-res → Enclosed-State

to target concept Emacs-Active yields target connection.

Emacs-Active → emacs-invoke-result* → Invoke-Emacs

Applying source path yields target concept Invoke-Emacs.

(A Invoke-Emacs2 (↑ Invoke-Emacs)
  (emacs-invoker2 (↑ emacs-invoker)
     (A I20 (↑ I)))
  (emacs-invoked2 (↑ emacs-invoked)
     (A Emacs12 (↑ Emacs))))
```

The system finds a hierarchical match between the source path relation `enter-res` and the target path concept `emacs-invoke-res`. Based upon this match, the concept `Invoke-Emacs`, attached to this relation, is chosen as the intended target concept. A new instance of this concept is created with the appropriate fillers from the primal representation.

```
 Mapping main source concept Entering to main target concept Invoke-Emacs.
  Mapping source role enterer to target role emacs-invoker.
  Mapping source role entered to target role emacs-invoked.

(A Entering-Invoke-Emacs (↑ Metaphor-Schema)
   (entered-emacs-invoked-map entered → emacs-invoked)
   (enterer-emacs-invoker-map enterer → emacs-invoker)
   (entering-invoke-emacs-map Entering → Invoke-Emacs))

Final interpretation of input:

(A How-Q18 (↑ How-Q)
   (topic18 (↑ topic)
    (A Invoke-Emacs2 (↑ Invoke-Emacs)
       (emacs-invoker2 (↑ emacs-invoker)
     (A I20 (↑ I)))
```

```
    (emacs-invoked2 (↑ emacs-invoked)
  (A Emacsl2 (↑ Emacs))))))
```

A full analysis of the nature and use of core-relationships for learning is given in Chapter 9.

6.5. The Extension Algorithm

The basic steps of the metaphor extension approach are outlined in the following sections. The details of each of these steps are then illustrated in terms of an annotated example. The full details of the Metaphor Extension System, which embodies the approach, are given in the next chapter.

Step 1: Characterize the new input. The new input consists of the primal representation for which the interpreter could not find an appropriate meaning. Partial source and target components of a new metaphor are extracted from this primal representation. The terms *current source* and *current target* will be used to refer to the source and target concepts derived from the input example.

Step 2: Search for related metaphors. This step searches for any known metaphors that are potentially related to this new use. The search consists of an attempt to find a path or paths through the KODIAK network from the current source concepts to the current target concepts through a known metaphor. A metaphor contained in such a path is judged to be relevant. The details of the search strategy will be given in the next chapter.

Step 3: Evaluate the set of candidate metaphors found in Step 2. The purpose of this step is to select a metaphor from the set found in Step 2 to be used as the basis for understanding the new use. This choice is based on a set of criteria to determine the metaphor that is closest conceptually to the current example. Two factors contribute to this conceptual distance. The first is the length of the core-relationship path from the source of the input concept to the source of the candidate metaphor. The second cost is the hierarchical distance from the target concepts of the candidate metaphor to the known target concepts of the input example. The metaphor with the shortest conceptual distance is chosen as the candidate metaphor.

Step 4: Apply the candidate metaphor found in the previous step to the input concepts. The candidate metaphor is applied to the current example based on its relationship to the current input concepts. The shape of the path connecting the primal concepts to the candidate metaphor indicates the type of inference that must be performed to extract the intended target meaning from the candidate metaphor. Depending on this path shape, a core extension inference, similarity extension inference, or a combined inference may be performed.

Step 5: Store the new metaphor. Once the intended target meaning for the current example has been determined, a new metaphor is created to represent this new use. A metaphor-sense is created whose source concepts are the source concepts extracted from the primal representation. The target concepts are the concepts found in Step 4. New metaphor-maps are created to associate each of these source target pairings. These new maps are then attached to the newly created metaphor-sense. The new metaphor-sense and its component metaphor-maps are then classified in the appropriate place in the metaphor hierarchy.

6.6. Annotated Example

The five steps of the extension system will now be illustrated by the following example. This example combines a core-related inference and a similarity inference.

```
> (do-sentence)

Interpreting sentence:

How can I get out of emacs?

Interpreting primal input.

(A Exiting2 (↑ Exiting)
   (agent50 (↑ agent) (A I3 (↑ I)))
   (patient50 (↑ patient) (A Emacs1 (↑ Emacs))))

Concreting input relations.

  Concreting  patient to exited.
  Concreting  agent to exiter.
```

The thematic relations specified by the parser are replaced here by the relations specific to the given primal concept.

```
Interpreting concreted input.

(A Exiting2 (↑ Exiting)
   (exiter2 (↑ exiter) (A I3 (↑ I)))
   (exited2 (↑ exited) (A Emacs1 (↑ Emacs))))

Failed interpretation: Exiting2 as Exiting.

No valid interpretations. Attempting to extend existing metaphor.
```

At this point, MIDAS has determined that there is no known coherent interpretation of the given primal input. It, therefore, appeals to its Metaphor Extension System in an attempt to extend an existing metaphor to explain this new use.

```
===========================================================
Entering Metaphor Extension System
===========================================================

Searching for related known metaphors.

Metaphors found: Enter-Lisp Enter-Association
```

Based on the characteristics of the primal input, the system initiates a search for relevant known metaphors. In this case, the system finds two known metaphors that share the concept `Entering` as a source concept. These metaphors are relevant by virtue of the fact that the source concept `Entering` is core related to the source concept `Exiting` of the new metaphor.

```
Candidate metaphor Enter-Association has ranking 8.
Candidate metaphor Enter-Lisp has ranking 4.

Selecting metaphor Enter-Lisp to extend from.
```

In the third step of the algorithm, the candidate metaphors are ranked according to their conceptual distance from the primal input concepts. In this case, the metaphor `Enter-Lisp` is selected for continued processing because of its closeness to the given inputs. The ranking of this metaphor results from a core-relationship path of length 2 from `Exiting` to `Entering`, and a conceptual distance hierarchical path from `Lisp` to `Emacs` through the common ancestor `Computer-Process` of length 2, resulting in a total ranking of 4.

```
Attempting a core-related metaphorical extension.

Extending similar core-related metaphor Enter-Lisp
with target concept Invoke-Lisp.

This is an intermediate extension inference.
```

In the next step, the system attempts to apply the new input example. In this case, the candidate metaphor is in a combined similarity and core-relationship to the primal input concepts. It is a core-related extension by virtue of the relationship between the candidate source concept `Entering` and the source concept of the new metaphor, Exiting. The similarity relationship arises from the hierarchical connection from the target concept Lisp in the candidate metaphor to the presumed target concept Mail in the new example.

```
Applying source path:

 Exiting → exit-pre → Enclosed-State → enter-res* → Entering

to target concept Invoke-Lisp yields target connection.

 Invoke-Lisp → lisp-invoke-result → Lisp-Active
→ lisp-uninvoke-pre* → Uninvoke-Lisp


Applying source path yields target concept Uninvoke-Lisp.
```

The first step in applying a combined core-related similarity inference is the application of the source core-relation path to the target domain to yield the appropriate target concept. In this case, the task is to apply the path connection from Entering to Exiting, to the target concept Invoke-Lisp of the candidate metaphor. Applying this path yields the concept Uninvoke-Lisp. At this point, the situation is the same as if the system had used an equivalent similarity related Exit-Lisp metaphor.

```
Abstracting Uninvoke-Lisp to ancestor concept Uninvoke-Comp-Process
producing abstract target meaning:

(A Uninvoke-Comp-Process1 (↑ Uninvoke-Comp-Process)
   (comp-process-uninvoked1 (↑ comp-process-uninvoked)
     (A Emacs1 (↑ Emacs)))
   (comp-process-uninvoker1 (↑ comp-process-uninvoker)
     (A I3 (↑ I))))
```

The MES now begins the combined abstraction-concretion inference designed to find a sibling or cousin concept to Invoke-Lisp that can accommodate the input constraints of the primal representation. In this example, it abstracts the target concept Uninvoke-Lisp to the concept Uninvoke-Comp-Process, which is the most specific ancestor of Uninvoke-Lisp that can accept the fillers in the primal representation. In particular, it can accept the filler Emacs in the role of the comp-process-uninvoked. At this point, an instantiation of the concept Uninvoke-Comp-Process1 is made with the appropriate fillers filled in. This is shown above as the concept Uninvoke-Comp-

Process1 with the filled in slots Emacs1 and I3.

```
Concreting Uninvoke-Comp-Process1 to concept Uninvoke-Emacs.

Yielding concept

(A Uninvoke-Emacs1 (↑ Uninvoke-Emacs)
   (emacs-uninvoker1 (↑ emacs-uninvoker)
     (A I3 (↑ I)))
   (emacs-uninvoked1 (↑ emacs-uninvoked)
     (A Emacs1 (↑ Emacs))))
```

The next phase in the application step is to find the most specific concept beneath this one that can accommodate the fillers to this concept. In this case, it is possible to concrete this general Uninvoke-Comp-Process to the more specific concept Uninvoke-Emacs. (This is actually the concept at which the answer to the users question is stored). Since the concretion phase selects this concept as the most specific concept available, the application step stops at this point.

```
 Mapping main source concept Exiting to main target concept Uninvoke-Emacs.
  Mapping source role exiter to target role emacs-uninvoker.
  Mapping source role exited to target role emacs-uninvoked.

(A Exiting-Uninvoke-Emacs (↑ Metaphor-Schema)
   (exited-emacs-uninvoked-map exited → emacs-uninvoked)
   (exiter-emacs-uninvoker-map exiter → emacs-uninvoker)
   (exiting-uninvoke-emacs-map Exiting → Uninvoke-Emacs))
```

Step 5 of the algorithm results in the creation of a new metaphor-sense to represent this newly understood use. Metaphor-maps are created linking the corresponding concepts in the source and target domains. These metaphor-maps are then tied to the new metaphor-sense via component-map relations. The newly created metaphor-maps and the metaphor-sense are then indexed into the metaphorical hierarchy in the appropriate places.

```
Final interpretation of input:

(A How-Q3 (↑ How-Q)
   (topic3 (↑ topic)
    (A Uninvoke-Emacs1 (↑ Uninvoke-Emacs)
       (emacs-uninvoker1 (↑ emacs-uninvoker)
     (A I3 (↑ I)))
       (emacs-uninvoked1 (↑ emacs-uninvoked)
     (A Emacs1 (↑ Emacs))))))
```

Since this is a UC question, the final representation of the user's question with the properly interpreted metaphor is passed along to UC for further processing.

```
Calling UC.

You can  get out of emacs by typing ^x^c to emacs.
```

Based on the previous example, the system has now acquired a new metaphor. This metaphor can be used directly in future metaphor interpretation and as a candidate for further extensions. In the following example, the system shows how this new metaphor can be applied directly during the interpretation process.

```
> (do-sentence)

Interpreting sentence:

How can I get out of emacs?

Interpreting primal input.

(A Exiting8 (↑ Exiting)
   (agent66 (↑ agent) (A I9 (↑ I)))
   (patient66 (↑ patient) (A Emacs7 (↑ Emacs))))

Concreting input relations.

  Concreting  patient to exited.
  Concreting  agent to exiter.

Interpreting concreted input.

(A Exiting8 (↑ Exiting)
   (exiter8 (↑ exiter) (A I9 (↑ I)))
   (exited8 (↑ exited) (A Emacs7 (↑ Emacs))))

Failed interpretation: Exiting8 as Exiting.

Valid known metaphorical interpretation.

Applying conventional metaphor Exiting-Uninvoke-Emacs.

(A Exiting-Uninvoke-Emacs (↑ Metaphor-Schema)
   (exited-emacs-uninvoked-map exited → emacs-uninvoked)
   (exiter-emacs-uninvoker-map exiter → emacs-uninvoker)
```

```
(exiting-uninvoke-emacs-map Exiting → Uninvoke-Emacs))
```

As in the previous example, the literal interpretation still fails. However, there is now a direct metaphorical interpretation whose source concept is attached to the primal input concept of `Exiting`. The direct application of this metaphor results in the creation of an instance of the `Uninvoke-Emacs` concept.

```
Final interpretation:

(A Uninvoke-Emacs7 (↑ Uninvoke-Emacs)
   (emacs-uninvoked7 (↑ emacs-uninvoked)
    (A Emacs7 (↑ Emacs)))
   (emacs-uninvoker7 (↑ emacs-uninvoker)
    (A I9 (↑ I))))
```

The final interpretation of the primal concept is assigned to its proper role in the representation of the rest of the user's utterance.

```
Final interpretation of input:

(A How-Q9 (↑ How-Q)
   (topic9 (↑ topic)
    (A Uninvoke-Emacs7 (↑ Uninvoke-Emacs)
       (emacs-uninvoked7 (↑ emacs-uninvoked)
     (A Emacs7 (↑ Emacs)))
       (emacs-uninvoker7 (↑ emacs-uninvoker)
     (A I9 (↑ I))))))
```

The newly interpreted question is then passed along to UC for further processing.

```
Calling UC.

You can get out of emacs by typing ^x^c to emacs.
```

6.7. Summary

The Metaphor Extension Approach is a method of learning new metaphors that uses the existing set of well understood metaphors as the main source of knowledge. The main thrust of the approach is to find a previously understood metaphor that is relevant to the new use that can be used to determine the meaning the intended target meaning of the new use. Two basic kinds of inferences are used in this extension process. These inferences are based upon two fundamental kinds of relationships in the system of known metaphors. Similarity extension inferences are used to extend existing metaphors that are judged to be hierarchically similar to the new use. Core extension inferences are used to extend existing metaphors to cover core-related new uses.

The next chapter describes the details of the Metaphor Extension System that embodies the approach outlined here. The details of Step 4, the step that actually finds the intended target meaning when given a candidate metaphor, will be discussed in Chapters 8 and 9. Chapter 8 gives the details of similarity learning. Chapter 9 gives the details of core-extension learning. Finally Chapter 10 presents a detailed series of examples and analyses from the previous literature.

Chapter 7

The Metaphor Extension System

7.1. Introduction

This Chapter describes the details of the learning component of MIDAS - the Metaphor Extension System (MES). The MES is called when, during the analysis of an input sentence, MIDAS can find no coherent conventional reading. The MES attempts to determine the meaning of this new use by extending an existing metaphor. Once the correct meaning has been determined, a new metaphor is created so that this new use can be recognized directly in future processing.

7.2. The MES Algorithm

In this section, I will first briefly describe the five steps of the algorithm. The detailed operation of each step will then be explained in terms of an example from the system.

Step 1: Characterize the new input. Partial source and target components of a new metaphor are extracted from the preliminary representation accepted as input from the interpreter. The terms *current source* and *current target* will be used to refer to the source and target concepts derived from the input example.

Step 2: Search for related metaphors. This step searches for any known metaphors that are potentially related to this new use. The search consists of an attempt to find a path or paths through the network from the current source to the current target concepts that contains a known metaphor. A metaphor contained in such a path is judged to be relevant.

Step 3: Evaluate the set of candidate metaphors found in Step 2. The purpose of this step is to select a metaphor from the set found in Step 2 for further processing. This choice is based on a set of criteria to determine the metaphor that is closest conceptually to the current example.

Step 4: Apply this previously understood metaphor to the current example. The candidate metaphor is applied to the current target based on the relationship between the candidate mapping and the current example.

Step 5: Store the new metaphor. Create and store a new metaphor consisting of the source and target concepts identified in the above steps and appropriate associations between them. This new metaphor will be used directly when future instances of this metaphor are encountered.

I will now step through a trace from MIDAS that will illustrate each of these steps. Consider the following example.

(1) John got the flu.

The situation is that MIDAS has no knowledge of any metaphors associated with the concept `Flu`. It does, however, have knowledge of the `Give-Cold` metaphor, as described in Chapter 4. In the following example, the system encounters Example (1), and learns the meaning of this new use through the use of the `Give-Cold` metaphor. This is an instance of a combined *similarity core-extension inference*. The similarity connection results from the close similarity connection between the concepts `Cold` and `Flu`. The core-extension relationship is through the common core concept, `Having`, shared by both `Getting` and `Giving`.

7.2.1. Step 0: Initial Processing

The interpreter parses the input sentence and determines that it can find no coherent conventional interpretation for this sentence. It, therefore, passes the primal representation along to the MES.

```
> (do-sentence)

Interpreting sentence:

John got the flu.

Interpreting primal input.

(A Getting18 (↑ Getting)
   (agent93 (↑ agent) (A John72 (↑ John)))
   (patient93 (↑ patient) (A Flu31 (↑ Flu))))
```

```
Concreting input relations.

  Concreting  patient to gotten.
  Concreting  agent to getter.

Interpreting concreted input.

(A Getting18 (↑ Getting)
   (getter18 (↑ getter) (A John72 (↑ John)))
   (gotten18 (↑ gotten) (A Flu31 (↑ Flu))))
```

MIDAS finds that neither the literal interpretation nor the single known metaphorical interpretation can meet the constraints posed by this primal representation.

```
Failed interpretation: Getting18 as Getting.
Failed interpretation: Getting18 as Get-Grade.

No valid interpretations. Attempting to extend existing metaphcr.
```

7.2.2. Step 1: Characterize The New Input

The input to the MES consists of the primal representation produced by the parser. The first step is to take this primal representation, and isolate the potential source and target components of this new metaphor.

Following the strategy laid out in Chapter 5, the main concept in the primal representation and its slots are taken to belong to the source of the new metaphor; the fillers of the slots are assumed to be concepts in the target domain. In this example, `Getting` along with its `getter` and `gotten` slots are taken to belong to the source domain. The concepts `John72` and `Flu31`, which are the fillers of the `getter` and `gotten` slots, are taken to be part of the target domain.

```
==========================================================
Entering Metaphor Extension System
==========================================================

(A Getting18 (↑ Getting)
   (getter18 (↑ getter) (A John72 (↑ John)))
   (gotten18 (↑ gotten) (A Flu31 (↑ Flu))))
```

7.2.3. Step 2: Collecting Relevant Metaphors

This step collects all the known metaphors that may be relevant to this new use. Known metaphors are judged relevant to the new example based upon their relationship to the current source and target concepts isolated in Step 1. This stage of the algorithm uses a rather loose set of criteria for determining relevance. The main idea is to use this loose criteria to quickly produce a list of candidate metaphors. This list may include metaphors of only marginal relevance that will be eliminated in subsequent processing.

A known metaphor is considered relevant if it satisfies one of the following criteria.

- The input source concept is the same as the source concept of the candidate metaphor.
- The input source concept is core-related to the source concept of the candidate metaphor.

The search begins at the input source concept, and spreads out to all the concepts core-related to the input source. Metaphors are added to the list of candidate metaphors as the search proceeds outward from the initial source concept.

The following four steps make up the basic search algorithm that selects the preliminary candidate set of relevant metaphors.

Search Algorithm

Step 1: Collect all the metaphors that have the input source concept as their source concept. This step simply collects all the metaphors that share the input source concept as a source.

Step 2: Find all the concepts that contain this concept as a core-metaphor, and collect their metaphors. If the input source concept is a core concept then all the concepts core-related to this core are examined, and their metaphors collected.

Step 3: Find all the concepts that are contained by this concept, and collect their metaphors. If the input source is not a core concept, but contains one or more core concepts, then they are found in this step.

Step 4 For each of these concepts found in step 3 find all the concepts that contain this concept. Collect the metaphors attached to these concepts. Step 3 basically shifts the search from an outlying concept to a core concept. This step then proceeds outward from the core to all the other core-related extensions to this core.

The key component of the search is the ability to find concepts that are core-related to the input source concept. The requirements for core-related concepts, laid out in

Chapter 3, form the basis for this search. The use of these restrictive requirements ensures that this search step will find relevant metaphors quickly. The three requirements for determining if two concepts are core-related are reproduced here from Chapter 3.

> **Relation Condition:** There is at least one relation whose aspectuals are constrained by the main concepts underlying each of the concepts.
>
> **Containing Condition:** All of the slots of one of the concepts are equated to slots of the other concept.
>
> **Shared Intermediate Condition:** Two concepts may be said to be core-related if they both satisfy the Relation and Containing conditions with a common third concept.

Recall that the key idea underlying these requirements is the notion that certain concepts serve as fundamental core concepts in the definitions of other concepts. These core concepts are said to be contained within the definitions of other concepts. For example, the concept of `Having` is at the core of the definition of the concepts `Giving` and `Getting`. Both are defined as actions or events that result in a `Having`. The concept `Having` is, therefore, a core concept, while `Getting` and `Giving` are core-related to it. The search algorithm uses the above representational requirements to guide the search for these core and core-related concepts.

In the current *get the flu* example, the search starts at the concept `Getting`. Step 1 yields the metaphor `Get-Grade`, which has the concept `Getting` as a source. (Note that for expository purposes the system has been deprived of its normal knowledge number of a number of other `Getting` metaphors. This was done to force the system to select a distant metaphor in order to demonstrate a wider range of search and application inference behavior). This metaphor is collected as a potential candidate metaphor. This connection is shown in Figure 1.

The second step yields no concepts that contain the concept `Getting` as a core concept. Step 3, however, finds that `Getting` contains the concept `Having` as a core concept. In this case, `Getting` and `Having` satisfy the Containing condition. As shown in Figure 2, the relation `getting-result` links the two concepts and the `haver` and `had` aspectuals of the `Having` concept are both equated to aspectuals of the `Getting` concept.

Examination of the `Having` concept finds that it is the source of the metaphors `Have-Permission` and `Have-State`. Therefore, these are added to the list of candidate metaphors. The state of the search after this step is shown in Figure 2.

In Step 4 the search proceeds out from the core concept `Having` to all the concepts that contain it as a core concept. (Excluding `Getting` which is where the search started.) At this point, the system finds that the concept `Giving` contains `Having` as a core concept and has two metaphors attached to it. The `Give-Cold` and `Give-Grade` metaphors are collected into the final list of candidate metaphors. Figure 3 shows the final extent of

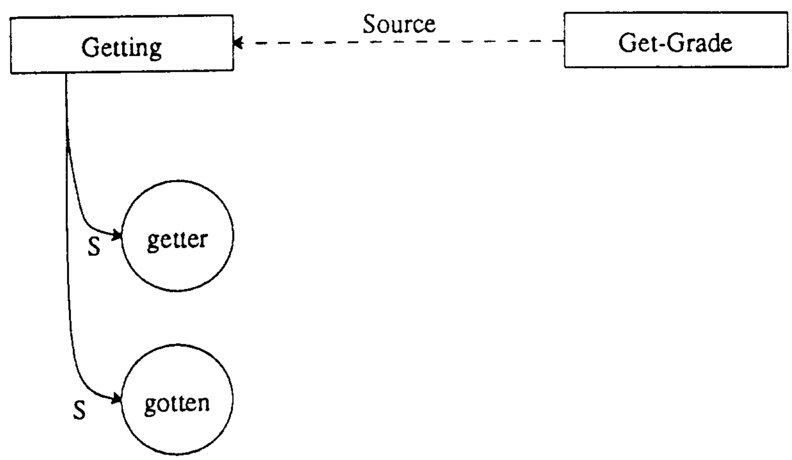

Figure 1: Search After Step 1

the search at the end of Step 4.

The search described here considers a metaphor to be relevant based solely upon the relationship of the source concept of the candidate to the input source concepts. This will clearly result in a number of irrelevant metaphors, since it ignores the input target concepts. In particular, it doesn't take into account the various kinds of similarity relationships among target concepts introduced in Chapter 3. This similarity information, provided by the target concepts, is taken into account in the next step of the algorithm, which evaluates the candidates and chooses the most appropriate one for continued processing.

```
Searching for related known metaphors.

Metaphors found: Get-Grade Have-Idea Have-Permission Have-State
Give-Cold Give-Grade
```

7.2.4. Step 3: Evaluating the Candidates

The next step in the algorithm is to select one of the metaphors found in the previous step for further processing. This selection represents the known metaphor that is most likely to yield the intended meaning with the least amount of effort.

In order to make this selection, the candidate metaphors are ranked based on their

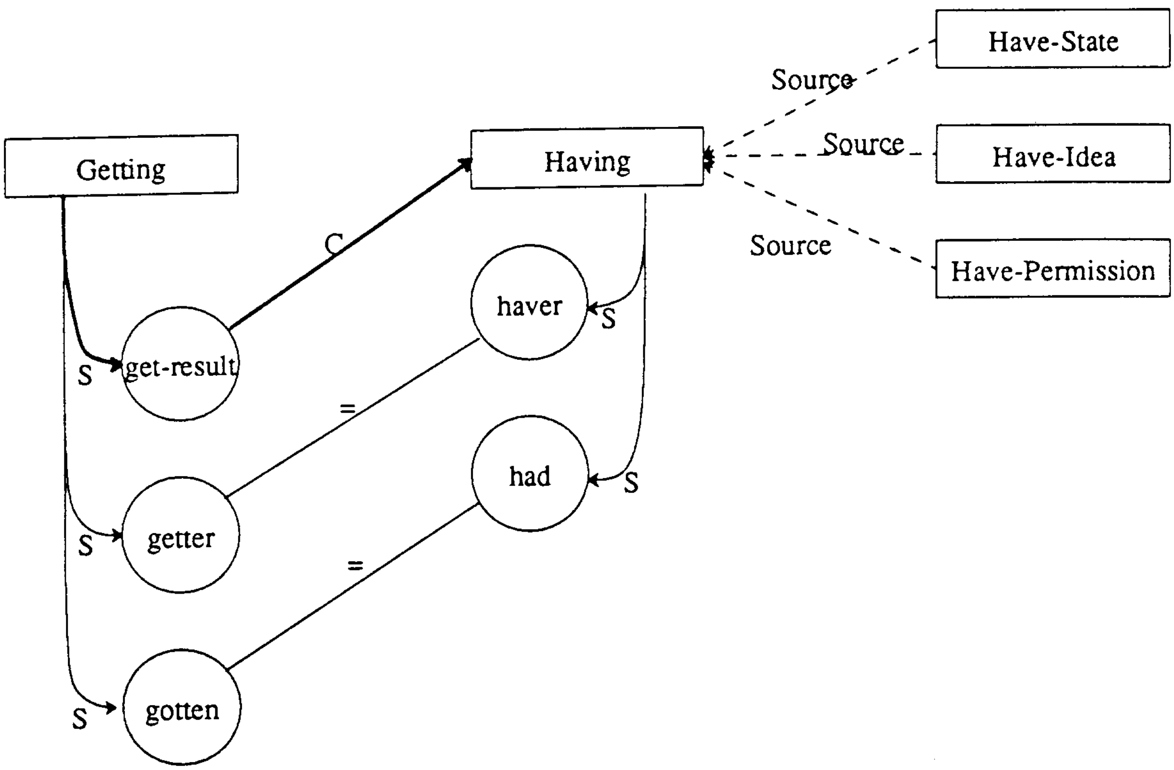

Figure 2: Search After Step 3

conceptual closeness to the input concepts specified in the primal representation. The basic approach taken is that the metaphor that is conceptually closest to the input concepts will be the easiest to map with the highest degree of certainty. Two factors contribute to this conceptual distance:

1) The length of the core-relationship from the input source to the source of the candidate metaphor.

2) A conceptual similarity measure indicating the distance from the input target concept to the target of the candidate metaphor.

The core-relationship component of the evaluation measure is simply the length of the core-relationship from the source of the candidate metaphor to the source concept of the new input metaphor. Each relation in the core-relation path counts as 1.

In the current example, the core-relationship from the input concept `Getting` to the source of the candidate metaphor `Giving` consists of the relations `getting-result` and `giving-result`. This path leads from the concept the `Getting` through the intermediate core concept `Having` to the concept `Giving`, yielding a path of length 2.

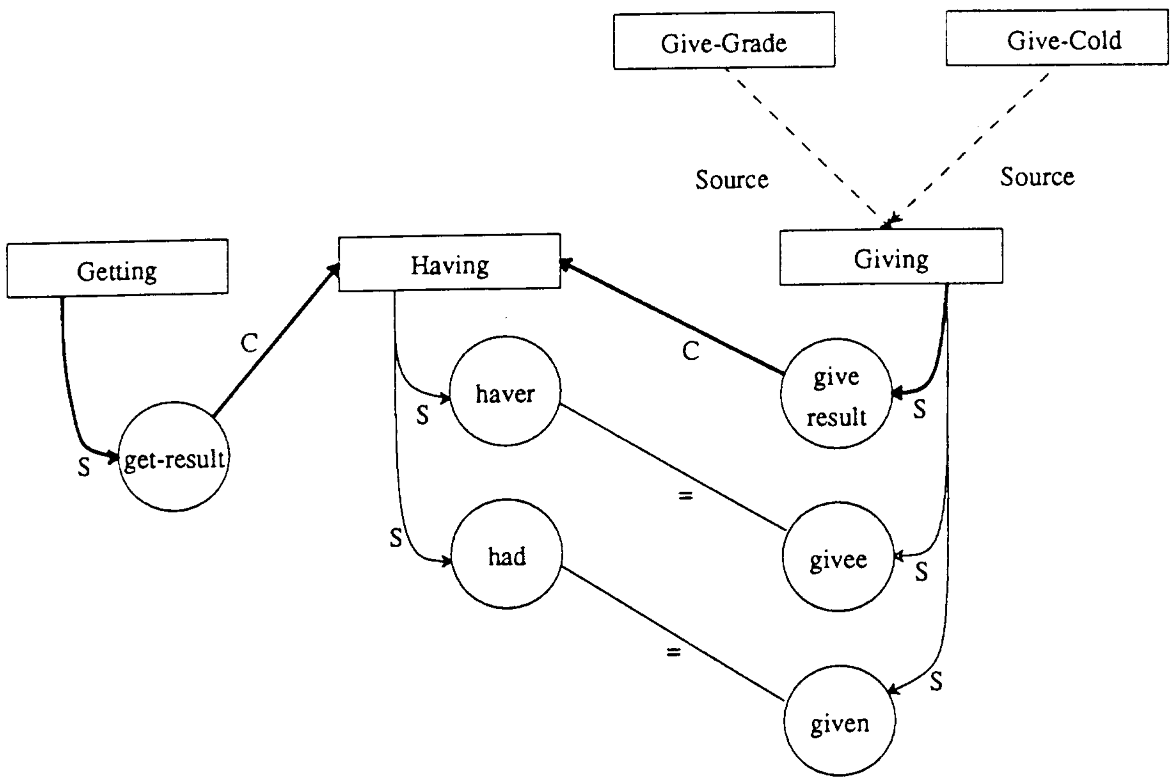

Figure 3: Search After Step 4

The conceptual similarity component of the evaluation measure is computed by measuring the hierarchical distance between the input target concepts and their corresponding target concepts in the candidate metaphor. The maximum distance from among all the input roles is taken as the conceptual similarity measure for that metaphor.

The hierarchical distance between two concepts is a measure of the distance between the concepts in the DOMINATE hierarchy. In the case where one of the concepts is an ancestor of the other, the hierarchical distance is simply the number of DOMINATE levels from the ancestor to the descendent. As discussed in Chapter 3, the typical metaphorical similarity relationship involves a sibling or cousin relationship. In these cases, the hierarchical distance is the sum of hierarchical distances from each of the concepts to their common ancestor.

Consider the candidate metaphor `Give-Cold`, chosen as the best candidate in the current example. The input target concepts of `John72` and `Flu31` fill the input roles of `getter` and `gotten`, respectively. Based upon the metaphor-maps of the candidate metaphor, these input roles correspond to the target concepts `cold-victim` and `infected-cold`. These target roles are constrained to be members of the categories `Person` and `Common-Cold`, respectively. The similarity measure of this metaphor is, therefore, based on the hierarchical distances from `Flu` to `Common-Cold` and from `John` to `Person`.

The common ancestor concept between the concepts `Flu` and `Common-Cold` is the concept `Disease`. This concept directly dominates both `Flu` and `Cold`, therefore, the total hierarchical distance between these concepts is 2. The remaining input role involves an ancestor rather than sibling relationship. The concept `Person` directly dominates the concept `John`, yielding the hierarchical distance from this role is 1. The final distance contributed by the hierarchical distance in this example is therefore 2.

The total conceptual distance for the `Give-Cold` metaphor is set to 4. The core-relationship contributes 2, and the hierarchical distance contributes 2. As shown below, this is the closest known metaphor from among all the relevant metaphors, and is passed along for further processing.

While the conceptual distance metric described here has worked well in practice, it does have a number of shortcomings. It relies heavily on the notion that conceptual closeness is directly reflected as distance in the abstraction hierarchy. This kind of measure is extremely sensitive to how the knowledge-base designer has chosen to arrange the levels in the hierarchy. This issue will be more fully discussed in Chapter 11, along with some related weaknesses of the system.

```
Evaluating candidate metaphor Get-Grade
  has a path length 0
  has a hierarchical distance 6
  giving ranking 6

Evaluating candidate metaphor Have-Idea
  has a path length 1
  has a hierarchical distance 4
  giving ranking 5

Evaluating candidate metaphor Have-Permission
  has a path length 1
  has a hierarchical distance 4
  giving ranking 5

Evaluating candidate metaphor Have-State
  has a path length 1
  has a hierarchical distance 5
  giving ranking 6

Evaluating candidate metaphor Give-Cold
  has a path length 2
  has a hierarchical distance 2
  giving ranking 4
```

```
Evaluating candidate metaphor Give-Grade
  has a path length 2
  has a hierarchical distance 6
  giving ranking 8

Candidate metaphor Get-Grade has ranking 6
Candidate metaphor Have-Idea has ranking 5
Candidate metaphor Have-Permission has ranking 5
Candidate metaphor Have-State has ranking 6
Candidate metaphor Give-Cold has ranking 4
Candidate metaphor Give-Grade has ranking 8.

Selecting metaphor Give-Cold to extend from.

(A Give-Cold (↑ Give-Metaphor Metaphor-Schema)
   (give-infect-res give-result → cold-inf-res)
   (given-infection given → infected-cold)
   (givee-infected givee → cold-victim)
   (giver-infector giver → cold-infector)
   (give-infect-map Giving → Cold-Infect))
```

7.2.5. Step 4: Applying the Candidate Metaphor

This step in the algorithm takes the candidate metaphor and applies it to the input target concepts in an attempt to find the intended target meaning. The method used to find this meaning is dependent upon the relationship between the candidate metaphor and the input concepts. The following two chapters will provide the full details of how the candidate metaphor is applied depending on the type of its relation to the new use. In particular, Chapter 8 will discuss the processing of similarity-related metaphors, and Chapter 9 will discuss core-related candidate metaphors. The rest of this section will provide an overview of how the MES can handle combinations of these extension processes.

The current *get flu* example is an instance of a combined core-extension and similarity inference. The candidate metaphor is core related to the inputs by virtue of the connection from the input source concept, `Getting`, to the source concept of the candidate target concept `Giving`. In addition, it is similarity related by virtue of the close similarity relationship between the input target concept `Flu` and the candidate target concept `Cold`.

The application of a combined core-similarity candidate metaphor is a two phase process in which the core-relationship is applied first. An abstraction-concretion inference is then applied to the result of the first phase. In terms of the current example, this strategy attempts to understand *get the flu* from the `Give-Cold` metaphor by first transforming *give a cold* to *get a cold* and then using that to find the meaning of *get the* *flu*. This strategy is effectively the same as if the system had first encountered and understood the ***get a cold*** metaphor in terms of the Give-Cold, and then subsequently had encountered the *get the flu* use.

```
Attempting a core-related metaphorical extension.

Extending similar core-related metaphor Give-Cold
with target concept Cold-Infect.

This is an intermediate extension inference.
```

At this point the MES has determined that the Give-Cold metaphor is a similarity and core-related candidate. In addition, it has determined, based upon the shape of the path, that the core-relationship between Getting and Giving is an intermediate extension type.

```
Applying source path:

 Getting → get-result → Having → give-result* → Giving

to target concept Cold-Infect yields target connection.

 Cold-Infect → cold-inf-res → Cold-Inf-State → bec-cold-res*
→ Bec-Inf-W-Cold

Applying source path yields target concept Bec-Inf-W-Cold.
```

The first step applies the source core-relationship to the target concept of the Give-Cold metaphor. This step yields the concept Bec-Inf-W-Cold, which basically represents the notion of becoming infected with a cold.

```
Abstracting Bec-Inf-W-Cold to ancestor concept Become-Infected producing
abstract target meaning:

(A Become-Infected7 (↑ Become-Infected)
   (infected-of-bec7 (↑ infected-of-bec)
    (A John72 (↑ John)))
   (infection-of-bec7 (↑ infection-of-bec)
    (A Flu31 (↑ Flu))))
```

The concept Bec-Inf-W-Cold is now abstracted to the concept Become-Infected based upon the constraints imposed by the input concept Flu. In this case, the concept

Become-Infected is the most specific ancestor of Bec-Inf-W-Cold that can accommodate the given inputs.

```
Concreting Become-Infected7 to concept Bec-Inf-W-Flu.

Yielding concept

(A Bec-Inf-W-Flu11 (↑ Bec-Inf-W-Flu)
   (bec-flu-victim11 (↑ bec-flu-victim)
    (A John72 (↑ John)))
   (bec-infected-flu11 (↑ bec-infected-flu)
    (A Flu31 (↑ Flu))))
```

Finally, the concept Become-Infected7 is concreted to the more specific concept Bec-Inf-W-Flu11. This is the most specific descendent of the Become-Inf concept that can accept the concept Flu in the infection-of-bec role.

7.2.6. Step 5: Storing the New Metaphor

This step produces a new metaphor representing a new use that can be used directly in future processing. Remember that the fundamental unit representing a coherent metaphor is the metaphor-sense. A metaphor-sense is a structured association consisting of a set of metaphor-maps that directly link the appropriate source and target concepts of the metaphor. The first task, therefore, is to create a new metaphor-sense with component metaphor-maps linking the various source and target concepts identified during the previous steps of the algorithm. The final step is to place these new metaphorical concepts at the correct level in the metaphor hierarchy.

In the current example, the system has determined that the following correspondences underlie this new metaphor: the main input concept Getting corresponds to Bec-Inf-W-Flu, the getter corresponds to bec-flu-victim, and the gotten corresponds to bec-infected-flu. Three new metaphor-maps, getting-bec-inf-w-flu-map, gotten-bec-infected-flu-map, and getter-bec-flu-victim-map are created to represent these metaphorical associations. The names of the maps (irrelevant to the functioning of the system) are created by concatenating the name of the source to the name of the target. These three maps are then attached to the newly created Getting-Bec-Inf-W-Flu metaphor-sense with component-map relations.

Note that two subtly different kinds of maps were actually created in this example. In the simple case, the metaphor-map getting-bec-inf-w-flu-map directly associates the main input source concept Getting with the main target concept Bec-Inf-W-Flu. This kind of simple metaphor-map, linking an absolute to an absolute, is called an absolute metaphor-map. The other maps, however, do not associate the target input filler concepts John and Flu with the corresponding source concepts. Rather, the new maps link the aspectual roles that input fillers play in the source domain with the corresponding roles in the target domain. This kind of map is called an aspectual map. This shift from the input concepts to the roles played by the inputs, represented as aspectuals, ensures that the metaphor-maps capture the correct metaphorical information presented in the example. It is the role of a person being infected with the flu and the role of a transmitted flu than that are being metaphorically structured, rather than the filler concepts John and Flu. The low-level KODIAK details of the newly created metaphor are shown in Figure 4.

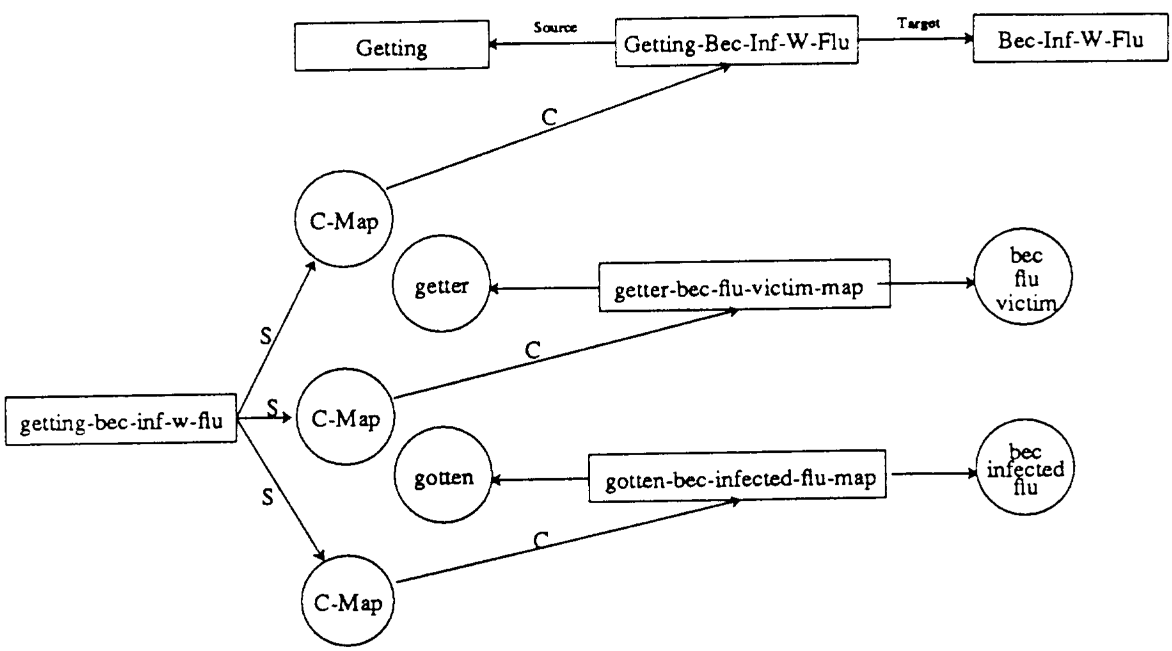

Figure 4: New *Get Flu* Metaphor

The final phase of this step in the algorithm is to place these newly created maps in the appropriate place in the hierarchy. A simple concretion process places each of the new maps under the most specific existing metaphor-map that can dominate the source and target concepts of the new map. The concretion simply starts at the top level concept `metaphor-map`, which represents a generic metaphor-map, and searches downward through the existing metaphor maps to classify the new maps. In the case of a simple absolute map, this classification is simply based upon the map's immediate source and target concepts. In the case of an aspectual map, the classification is based not upon the source and target concepts of the metaphor-map (which are aspectuals) but upon the constrainers of these aspectual concepts. Note that by uniformly representing metaphor-maps as ordinary KODIAK concepts, it is possible to use the existing concretion mechanisms to classify these new concepts.

```
Mapping main source concept Getting to main target concept Bec-Inf-W-Flu.
 Mapping source role getter to target role bec-flu-victim.
 Mapping source role gotten to target role bec-infected-flu.

(A Getting-Bec-Inf-W-Flu (↑ Metaphor-Schema)
   (gotten-bec-infected-flu-map gotten → bec-infected-flu)
   (getter-bec-flu-victim-map getter → bec-flu-victim)
   (getting-bec-inf-w-flu-map Getting → Bec-Inf-W-Flu))
```

7.3. Summary

This chapter has presented the details of the learning component of MIDAS - the Metaphor Extension System. The MES accepts a representation of a new use for which no coherent conventional interpretation was possible. It then attempts to infer and remember an appropriate interpretation of this use by systematically extending an existing metaphor. This chapter has provided the details for how the system finds and selects an appropriate candidate metaphor, and also how the system stores the newly understood metaphor for future use. The exact details of the similarity-extension and core-extension inferences, which constitute Step 4 of the MES algorithm, will be given in the next two chapters.

Chapter 8

Learning Metaphors by Using Similarity

8.1. Introduction

This chapter describes how a new metaphor can be understood by reference to a known conventional metaphor that has been judged to be similar to the new use. As described previously, understanding a new metaphor principally means determining the concept that plays the role of the target in this new metaphor. The similarity approach relies on the fact that this new target concept may be similar to a concept that is the target meaning of some already understood metaphor. It should be possible, therefore, to determine the meaning of the new metaphor by mapping from the meaning of an existing similar one.

The strategy just described infers further similarities between concepts based on some limited set of known similarities. This general strategy has received a great deal of attention under the various headings of *Analogical Reasoning* (for recent surveys see Pridietis 1987 and Helman 1988) and *Case-Based Reasoning* (Hammond 1986). The approach described in this chapter applies this general strategy to the specific task of understanding new metaphors. It asserts that there are certain domain specific similarities in conventional metaphors that are strongly predictive of further similarities. Note that the approach described here is not directly related to that known as *Similarity-Based Learning* (Michalski 1983). Similarity-Based Learning is concerned with the induction of a general concept based on a series of positive and negative input examples.

Consider the following examples.

(1) John gave Mary a cold.
(2) John gave Mary the flu.

Consider the situation where an understander has knowledge of a highly specific metaphor underlying Example (1). No generalized metaphor has yet been formed for this class of metaphor. Example (2) is then encountered. The hierarchical relationship between colds and flu can be exploited to understand this new use by assuming that the target meaning of the known use has a corresponding meaning with the new concept. In this example, since *give* means to infect when applied to colds, it will probably mean *infect* when applied to the flu, since colds and flu are known to be similar concepts.

Exploiting the abstraction hierarchy is the key to using a similar metaphor to understand a new use. The basic strategy is to use the given input target concepts to guide a search through the hierarchy from the target of the known metaphor to the intended target of the new use. This search proceeds from the target concept of the known metaphor to a more abstract concept, and then down to the most specific concept that can accept the input concepts. This abstraction and specialization search is a result of an inference called a *combined abstraction-concretion* inference. This inference will be illustrated with an example in the next section, and defined fully in the algorithm section.

The focus in this chapter is on the nature of the similarity relationship and how it can be exploited once an appropriate metaphor has been identified. This corresponds to Step 4, the application step, of the MES algorithm.

8.2. An Example

This section will consider the details of the situation where a listener has knowledge of the highly specific `Give-Cold` metaphor, as described in Chapter 4, and reproduced here in Figure 3. In addition, the listener has knowledge of other infectious diseases like the flu. The listener, however, has never encountered the use of this metaphor with any disease other than the common-cold. The question is how can such a listener come to an understanding of (2) when it is encountered. In particular, how can the closeness of this new use to the `Give-Cold` metaphor, and the information available in the example, be used to identify the concept `Infect-With-Flu` as the intended target meaning?

This step of the overall MES algorithm takes the primal input and a similarity related candidate metaphor and finds the intended target meaning. Figure 1 shows the primal representation of Example (2). MIDAS has not been able to find any conventional interpretation that can adequately accommodate this input. The MES has, in turn, accepted this input, and during the search and evaluation steps decided that the `Give-Cold` metaphor is the most closely related candidate. Figure 2 shows the connections between the primal input concepts and this candidate metaphor. The `Give-Cold` metaphor itself is show in Figure 3.

Figure 4 shows the hierarchical connection between the target concepts of the `Give-Cold` metaphor and the intended target concept of the flu example. In this diagram, the target meaning of (1) is shown as the `Infect-With-Cold` concept with the `infection` constrained to be a `Common-Cold`. The intended meaning of (2) is shown as the `Infect-With-Flu` concept with the infection specified as an instance of `Flu`. These concepts are in turn dominated by common parent categories. In particular, `Infect-`

```
(A Giving1 (↑ Giving)
    (giver1 (↑ giver) (A John1 (↑ John)))
    (givee1 (↑ givee) (A Mary1 (↑ Mary)))
    (given  (↑ given) (A Flu1 (↑ Flu))))
```

Figure 1: Primal Representation

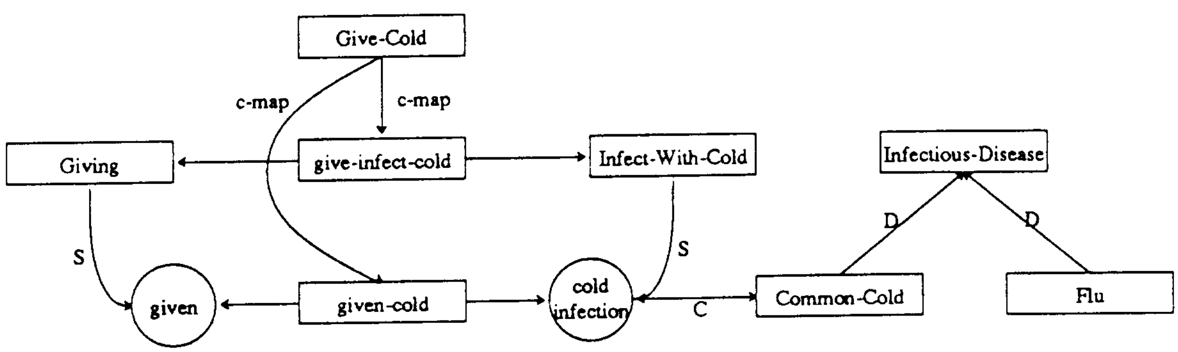

Figure 2: Give-Cold Similarity Connection

With-Cold and Infect-With-Flu are dominated by the Infect-With-Disease concept. Correspondingly, the Flu and Common-Cold concepts are dominated by the common parent Infectious-Disease. Figure 5 shows the more complete relationships associated with the parent concepts Infect-With-Disease and Infectious-Disease.

The key to using a known similar metaphor to understand a new use is being able to move through the hierarchy from the known target concept to the intended target concept. This search is guided by the concepts that have been identified as the partial target concepts. Briefly, this search is a two step process that first proceeds from the target of the candidate metaphor to an appropriately abstract ancestor concept, and then down to the appropriate target via a concretion inference.

The first step in processing this example is to pair up the input target concepts with their corresponding concepts in the candidate target concept. This is accomplished by pairing the fillers of the input source roles with the target concepts of the candidate metaphor. In this case, the filler of the giver role, John1, is paired up with the target concept cold-infector, in accordance with the giver-infector metaphor-map of the Give-Cold metaphor. In a similar manner, the filler concepts of the source roles givee and given, Mary1 and Flu1, are paired up with the candidate target roles cold-

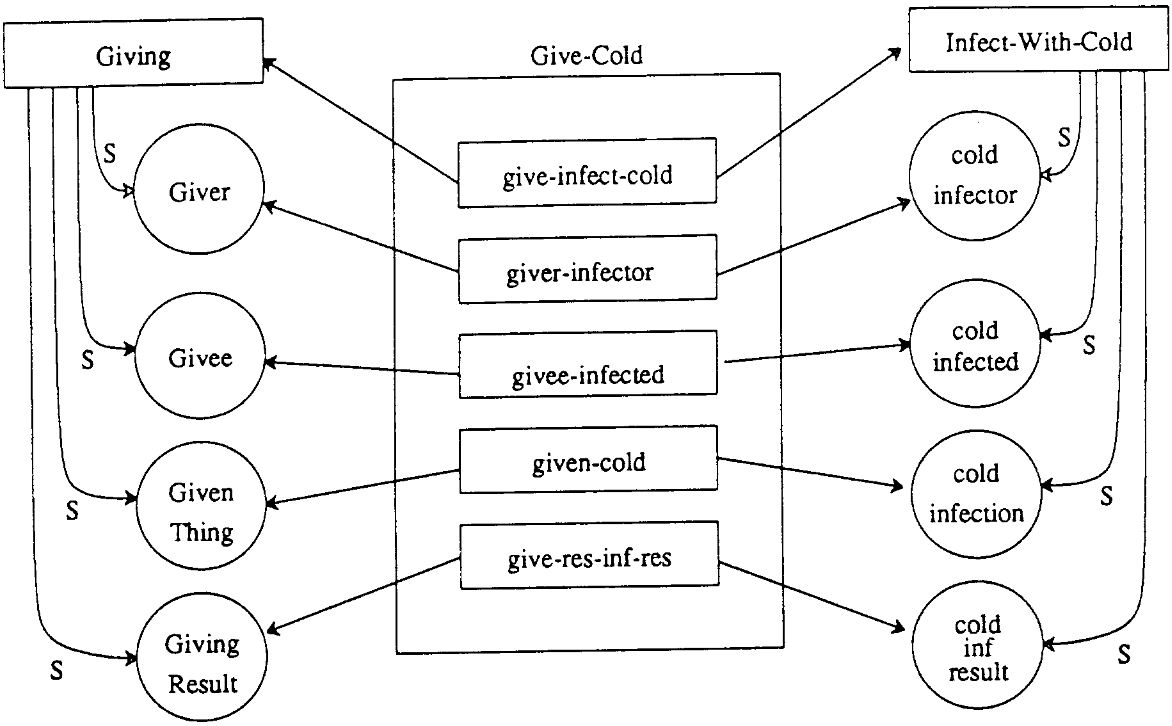

Figure 3: Give-Cold Metaphor

infected and cold-infection.

The next step in the processing is to use these pairings to guide the search for an appropriate abstraction of the target candidate metaphor. The search begins at the target concept of the known metaphor, Infect-With-Cold, and moves upward through the hierarchy to the more abstract concept Infect-With-Disease. This upward movement is guided by the pairings of the input concepts with candidate target roles. The search proceeds upward until a concept is found whose constraints are abstract enough to accept the given fillers. In this case, this abstract concept, Infect-With-Disease, is the immediate parent of the target concept Infect-With-Cold. As shown in Figure 5, the constraints on the slots of this concept can all accept the given input filler concepts in the appropriate roles. In particular, John1 and Mary1 can fill the abstract aspectuals, infector and infected, since they are constrained by the concept Person. Similarly, the input filler, Flu1, can fill the abstract role of infection, which is constrained to be an Infectious-Disease.

The result of this initial abstraction step is the assumption that the intended target meaning is a kind of Infect-With-Disease, where John1 plays the role of the infector, Mary1 plays the role of the infected, and Flu1 plays the role of the infection. This abstract concept represents a plausible interpretation of the inputs since it can coherently account for all the input concepts. It may, however, not be the most specific

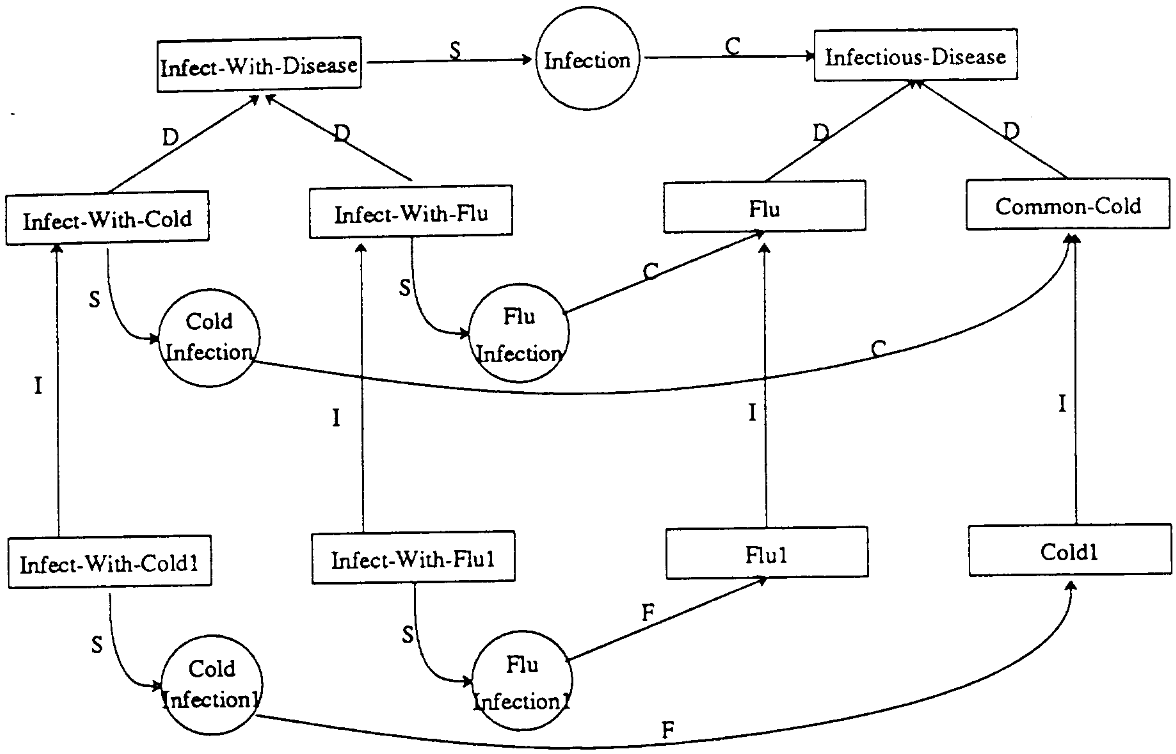

Figure 4: Cold and Flu Infecting

known concept that can account for the inputs. In particular, there may be a concept that represents specific knowledge about infecting someone with the flu. The next step is an attempt to find a concept more specific than this abstract concept.

The next step is simply a concretion inference. The concept `Infect-W-Disease`, with the fillers assigned to the appropriate roles, is concreted to the most specific concept below it that can accept these input concepts. This concretion leads to the concept `Infect-With-Flu`, which has `Infect-With-Disease` as an immediate parent. This concept is accepted because the constraint on its version of the `infection` slot is more specific than the constraint on the parent concept, yet it can still accept the input filler concept `Flu1`.

Note that the prior existence of the specific `Infect-W-Flu` concept is not necessary to the success of this process. In the event that there is no known concept more specific than `Infect-With-Disease`, the system creates a new specific instance of `Infect-With-Disease` with the concept `Flu` constrained to play the role of the infecting disease. This new instance plays the role of the target concept of the new metaphor.

The process of mapping a closely related metaphor is, therefore, one of abstracting

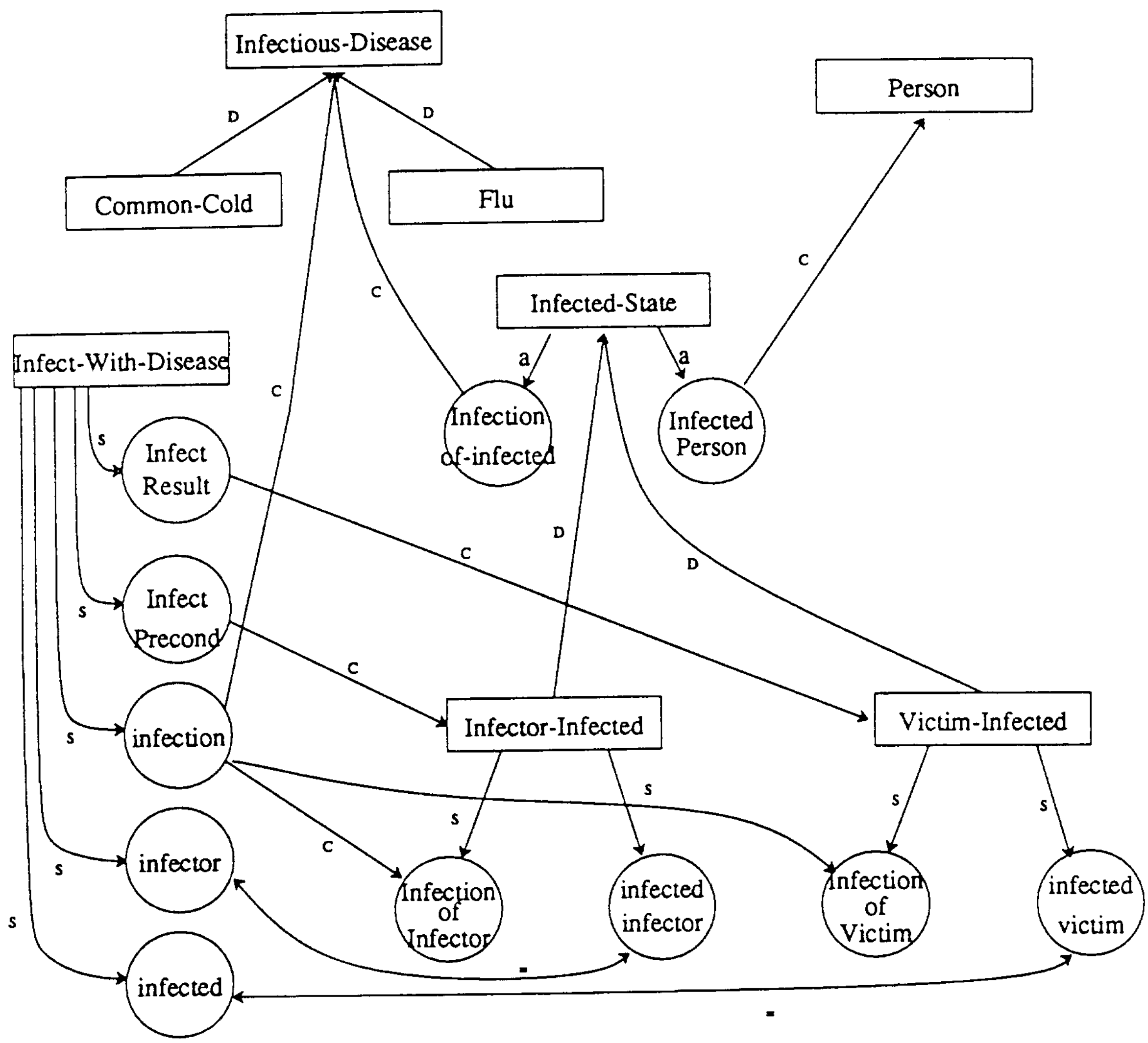

Figure 5: Infecting

the main target concept of the known metaphor to an abstract enough level to accept the inputs, and then attempting to specialize* it down to the most specific concept that can accept the given input constraints.

* Alterman (1988) employs a similar abstraction-specialization algorithm to perform the task of adapting a known plan to a new problem solving situation.

8.3. Similarity Algorithm

Now that we have seen a simple example of similarity extension inference, this section will present the exact details of the similarity extension algorithm. Once each of the steps has been described, a detailed trace example from the MES will be walked through.

The algorithm described here corresponds to the application step (Step 4) of the overall MES in those cases where the candidate metaphor is similarity-related to the new metaphor.

Step 0: Accept a candidate metaphor and primal representation. The MES has been passed a primal representation for which there is no adequate conventional explanation. The MES, as a result of its first two steps, finds and selects a similarity related candidate metaphor.

Step 1: Assign corresponding roles. In this step the target concepts in the primal representation are paired up with the corresponding concepts in the target concept of the candidate metaphor. This pairing is based upon the common roles that the target concepts play in the source domain.

Step 2: Abstract the target concept. The target concept of the candidate metaphor is replaced by its most specific ancestor that can accommodate the given input concepts.

Step 3: Concrete the abstract target. Once an abstract concept has been found it is then replaced by its most specific descendent that can accept the given input concepts.

These steps will be made more concrete by considering the detailed processing of the following examples:

(3) John killed the conversation when he arrived.
(4) How can I kill a process?

Consider the situation where (4) is encountered for the first time and the system attempts to use the `Kill-Conversation` metaphor underlying (3) to understand the new use. This example illustrates a more distant kind of similarity than was seen in the previous one. In particular, the relationship between the target concepts, conversations and computer-processes, is much more distant. The abstraction step of the algorithm may be forced to ascend to an arbitrary height in the hierarchy before a concept is found that is abstract enough to accept the input concepts. The distance traveled reflects how close the candidate metaphor actually is to the current example.

8.3.1. Step 0: Accept

Consider the situation where example (4) has been encountered and the `Kill-Conversation` metaphor has been selected as the best candidate metaphor. Figure 6 shows the primal representation given to the MES. Based on this input, the MES has selected the known `Kill-Conversation` metaphor as the best candidate to work from. This metaphor is shown in Figure 7.

```
(A Killing1 (↑ Killing)
    (killer1 (↑ killer) (A I1 (↑ I)))
    (kill-victim1 (↑ kill-victim)
       (A Computer-Process1 (↑ Computer-Process))))
```

Figure 6: Primal Representation

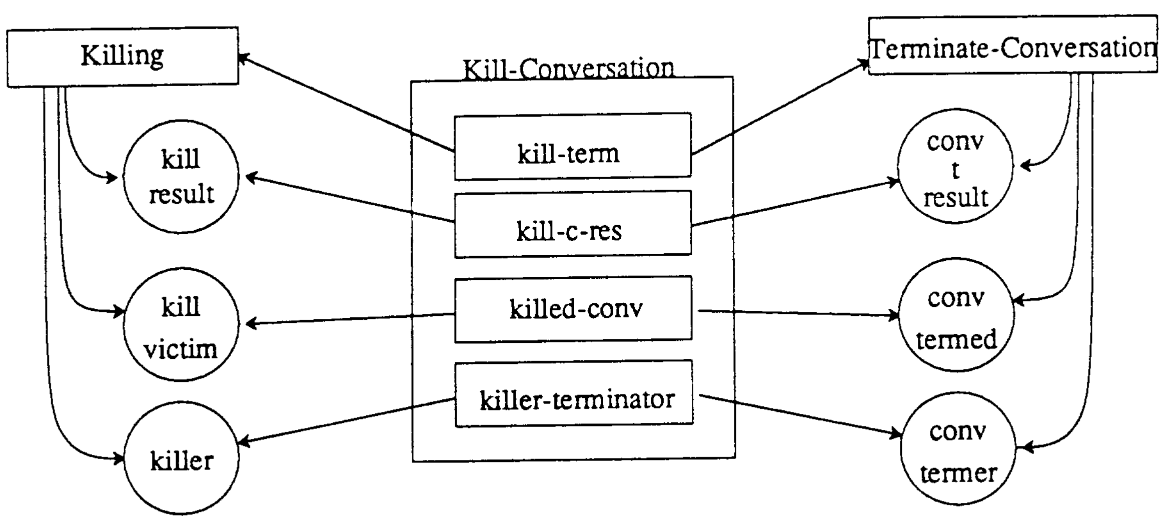

Figure 7: Killing a conversation

The target meaning of (3), shown in Figure 8, consists of the concept `Terminate-Conversation` which is an action that has an effect that causes the `terminated-conversation` to be in a `Terminated-Conversation-State`. The `terminated-conversation` is in turn constrained to be a `conversation`. The dashed DOMINATE links are an abbreviation indicating that the dominating category shown is an eventual parent of the child, but does not directly dominate it.

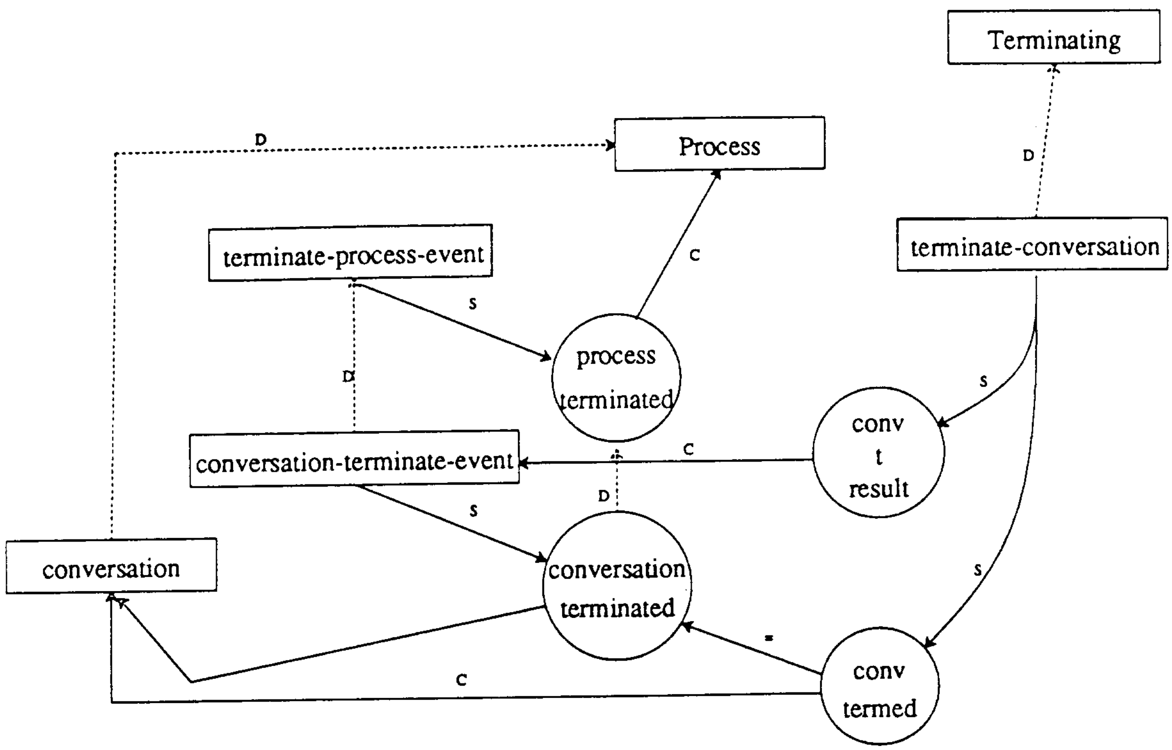

Figure 8: Terminating a conversation

```
> (do-sentence)

Interpreting sentence:

How can I kill a process?

Interpreting primal input.

(A Killing16 (↑ Killing)
   (agent87 (↑ agent) (A I46 (↑ I)))
   (patient76 (↑ patient)
    (A Computer-Process10 (↑ Computer-Process))))

Concreting input relations.

  Concreting  patient to kill-victim.
  Concreting  agent to killer.
```

```
Interpreting concreted input.

(A Killing16 (↑ Killing)
   (killer16 (↑ killer) (A I46 (↑ I)))
   (kill-victim16 (↑ kill-victim)
    (A Computer-Process10 (↑ Computer-Process))))

Failed interpretation: Killing16 as Killing.
Failed interpretation: Killing16 as Kill-Delete-Line.
Failed interpretation: Killing16 as Kill-Sports-Defeat.
Failed interpretation: Killing16 as Kill-Conversation.

No valid interpretations. Attempting to extend existing metaphor.

===========================================================
Entering Metaphor Extension System
===========================================================

Searching for related known metaphors.

Metaphors found: Kill-Conversation Kill-Delete-Line Kill-Sports-Defeat

Selecting metaphor Kill-Conversation to extend from.

(A Kill-Conversation (↑ Kill-Metaphor Metaphor-Schema)
   (kill-c-res kill-result → conv-t-result)
   (killed-conv kill-victim → conv-termed)
   (killer-terminator killer → conv-termer)
   (kill-term Killing → Terminate-Conversation))
```

8.3.2. Step 1: Assign Roles

The first step pairs up the input target concepts with their corresponding concepts in the candidate target. This is accomplished through the use of the source roles and the metaphor-maps in the candidate target metaphor. In this example, `I46`, which plays the role of the `killer` in the source, is paired up with the `conv-termer` role in the candidate target. The `conv-termer` concept is chosen because it is attached to a metaphor-map in the candidate metaphor that has the concept `killer` as a source. Similarly, the concept `Comp-Process10`, which plays the role of the `kill-victim` in the source, is paired up with the `conv-termed` in the candidate target role.

```
Attempting a similarity extension inference.

Extending similar metaphor Kill-Conversation with target concept
Terminate-Conversation.

Assigning input concept I46 to candidate target role conv-termer.
Assigning input concept Comp-Process10 to candidate target role
conv-termed.
```

8.3.3. Step 2: Abstraction

The next step is to abstract the target concept of the candidate metaphor to an appropriately abstract concept. This abstract concept is the most specific ancestor of the candidate target that can accommodate the input concepts in the roles assigned to them in Step 1. This abstraction is actually accomplished by moving upwards through the ancestors of the target aspectuals themselves. The search stops at the lowest concept where all of the aspectuals can be accepted.

In this example, each of the target aspectuals, `conv-termer` and `conv-termed`, are abstracted in turn. It is immediately found that since the concept `conv-termer` is constrained to be a `Person`, the input concept `I46` can be accepted in that role. However, the concept `Computer-Process10` cannot fit the role of `conv-termed` since it is constrained to be a `Conversation`, and `Computer-Process10` is not dominated by this concept. Therefore, the abstraction process must move upward. It proceeds upward through the ancestors of `conv-termed` until the concept `terminated` is found. This concept is constrained to be the abstract concept `Process`, which is an ancestor of `Computer-Process10`. The concept `terminated` is an aspectual of the concept `Terminate`, which will now be considered as an abstraction of the intended target meaning.

Figure 9 shows the hierarchical relationship between the target concepts, `Conversation` and `Computer-Process`. The common ancestor ultimately dominating these concepts is `Process`. This represents the abstract notion of a series of actions and events occurring over time. A `Conversation` is a sequence of speech acts by two or more parties occurring over some time period, serving a particular social function. A `Computer-Process` consists of a series of directed computer operations over some time period.

The termination actions underlying the target meanings of these examples have a corresponding hierarchical relationship. The exact details of how a conversation is halted and how a process is halted are very different concepts. They do, however, share a common core notion of bringing an ongoing sequence of events to a halt.

The final step of this abstraction process is the instantiation of a new instance of the `Terminating` concept with the input concepts `I46` and `Computer-Process10` assigned to the abstract roles of `terminator` and `terminated`, respectively.

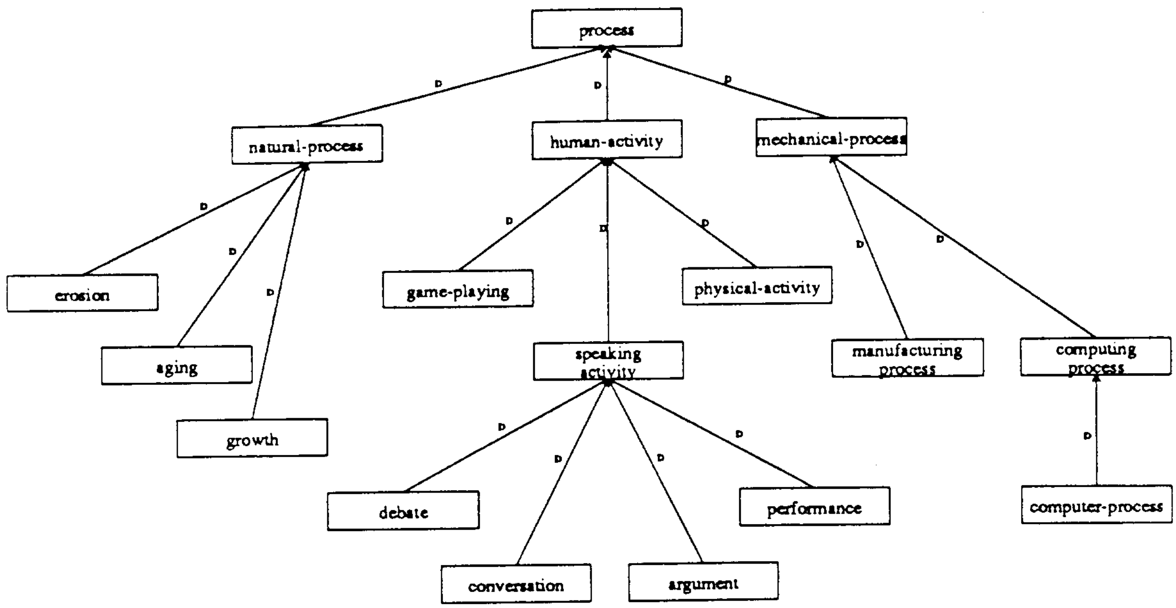

Figure 9: Process Conversation Connection

```
Abstracting Terminate-Conversation to ancestor concept Terminating
producing abstract target meaning:

(A Terminating3 (↑ Terminating)
   (terminated3 (↑ terminated)
    (A Computer-Process10 (↑ Computer-Process)))
   (terminator3 (↑ terminator) (A I46 (↑ I))))
```

The result of the abstraction step is extremely sensitive to the closeness of the candidate metaphor to the input examples. In Example (5), discussed above, the abstract category found was the immediate parent of both the candidate target concept and the intended target meaning. In the current example, the abstract concept `Terminate` is a further distance away in the hierarchy than just the immediate parent of the given target concept. Section 8.4, below, will discuss the general issue of degree of similarity and situations where the concept found in this step is too abstract to be easily usable.

8.3.4. Step 3: Concretion

The final step is to concrete the abstract concept, found in Step 2, down to its most specific descendent that can accept the given input concepts in more specific roles. As in the case of the abstraction step, the concretion is entirely based upon finding concepts whose aspectuals can accept the input concepts. The search, therefore, proceeds downward through the descendents of the abstract aspectuals `terminated` and `terminator`. This concretion inference yields the concept `Terminate-Computer-Process`. This is the most specific descendent of the concept `Terminating` that can accept the input concepts. In particular, it can accept the concept `I46` in the role of the `c-proc-termer`, which is a kind of `terminator`, and `Computer-Process10` in the role of the `c-proc-termed`, a kind of `terminated` that is constrained to be a `Computer-Process`.

```
Concreting target concept Terminating to Terminate-Computer-Process
producing concreted meaning:

(A Terminate-Computer-Process10
   (↑ Terminate-Computer-Process)
   (c-proc-termer10 (↑ c-proc-termer)
    (A I46 (↑ I)))
   (c-proc-termed10 (↑ c-proc-termed)
    (A Computer-Process10 (↑ Computer-Process))))


Creating new metaphor:

 Mapping main source concept Killing to main target concept
 Terminate-Computer-Process.
  Mapping source role killer to target role c-proc-termer.
  Mapping source role kill-victim to target role c-proc-termed.

(A Killing-Terminate-Computer-Process (↑ Kill-Metaphor)
   (kill-victim-c-proc-termed-map kill-victim → c-proc-termed)
   (killer-c-proc-termer-map killer → c-proc-termer)
   (killing-terminate-computer-process-map
       Killing → Terminate-Computer-Process))

Final interpretation of input:

(A How-Q46 (↑ How-Q)
   (topic46 (↑ topic)
    (A Terminate-Computer-Process10
      (↑ Terminate-Computer-Process)
      (c-proc-termer10 (↑ c-proc-termer)
     (A I46 (↑ I)))
      (c-proc-termed10 (↑ c-proc-termed)
     (A Computer-Process10
```

(↑ Computer-Process))))))

Calling UC.

You can kill a computer process by typing ^c to the shell.

Figure 10 shows the meaning underlying (4). The main concept is the `Terminate-Computer-Process` action. This is an action that causes a running process, the `terminated-computer-process`, to be in a `Terminated-Computer-Process-State`.

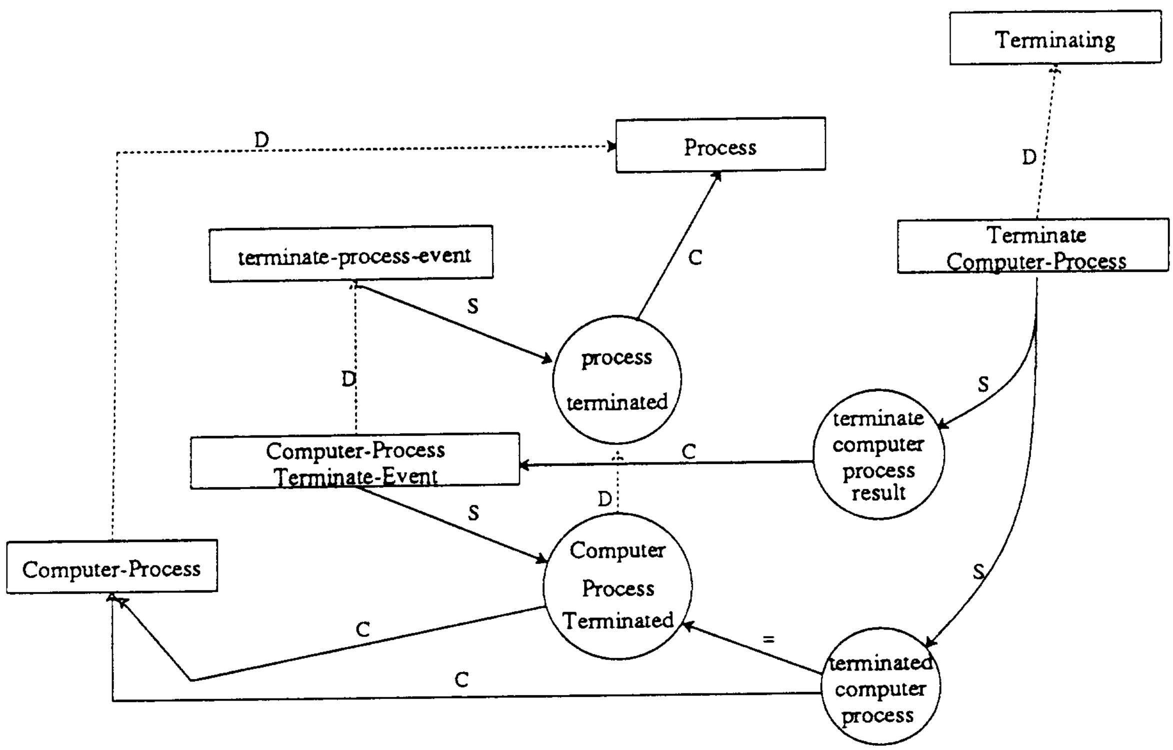

Figure 10: Terminating a computer-process

In general, the difficulty of the concretion step is completely dependent upon the kind of abstraction found in Step 2. In particular, the higher the level of abstraction of the concept found in Step 2, the more difficult the concretion becomes. As mentioned above, this is a function of how close the candidate metaphor is to the current example. In the current example, there is a unique concept beneath `Terminating` that has the concept `Computer-Process` in the role of `terminated` the concretion is, therefore, relatively easy.

The combined abstraction-concretion strategy described here is a form of analogical reasoning. The known metaphor provides a salient analog in its target concept. This analog, along with the partial target concepts provided in the primal representation, are used to find a concept analogous to the target of the known metaphor that is consistent with the given input concepts. The success of this strategy is completely dependent upon how good the specified analog concept is.

8.4. Degrees of Similarity

It is obvious from the previous example that the success of the similarity approach depends critically upon the relationship between the candidate metaphor and the new example. It is important to understand the exact role that similarity and abstraction play in this process. As shown in the *Kill* example, the target concepts in the candidate metaphor may be quite distant from the new example and still yield useful results. The critical factor is not the degree of similarity between the target concepts of the new and candidate metaphor. Instead, the critical factor is whether or not there exists a common abstraction between these concepts that is meaningful with respect to the new metaphor. Consider the following examples.

(6) Mary gave John an idea.
(7) Mary gave John a cold.

These examples are in an even more distant relationship than the previous example involving *kill*. The similarity in these examples is more a result of the constraints on the metaphors, rather than of the semantics of the target concepts themselves. This is illustrated by the fact that, unlike the previous examples, the common abstraction between the target concepts in these examples is highly abstract. Figure 11 shows the details of the target concepts underlying (6). Figure 12 shows the distant hierarchical relationship between the target concepts, `Infect-With-Cold`, introduced earlier, and the target concept underlying (6), `Induce-Idea`.

The common ancestor of these concepts is the abstract notion of causing someone to be in a new state with respect to some previously unrelated concept. This concept is shown as the concept `Cause-New-State` in Figure 12. The concept `Infect-With-Cold` is, therefore, a kind of `Cause-New-State` where the victim of the infection enters a new physical state with respect to a cold infection. Similarly, `Induce-Idea` is an action that results in the effected person being in a new mental state with respect to a particular idea. The concept `Cause-New-State` is, therefore, an abstract concept that dominates more specific concepts that may have physical, mental, or emotional effects on the effectea person. Specific information about the nature of the actions and effects lies almost completely in these more specific concepts.

The difference between this kind of distant hierarchical relationship and that found

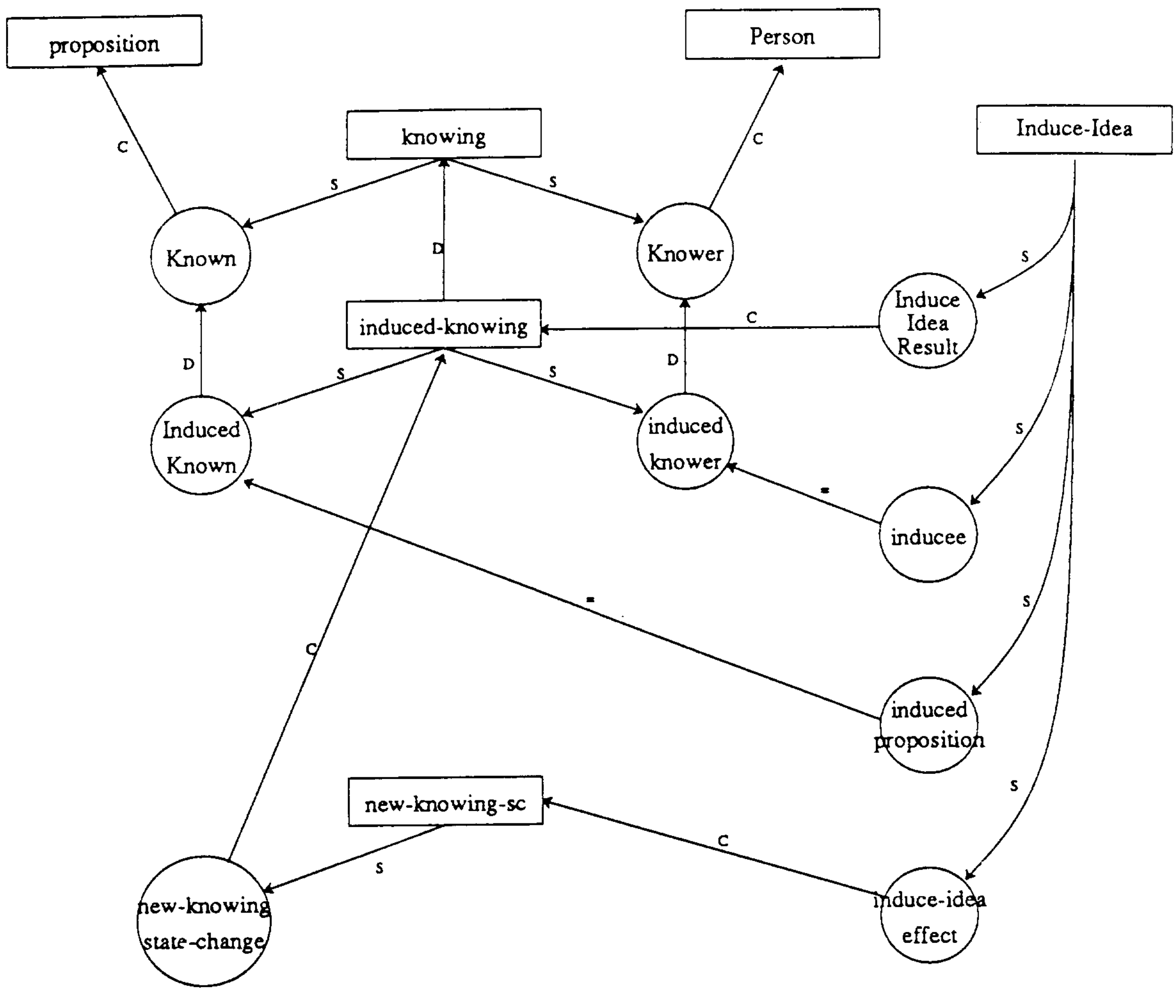

Figure 11: Induce an idea in a person

in the *Kill* example is one of degree. Contrast the concept `Cause-New-State` with the common abstract concept `Terminate` in the *Kill* examples. The `Terminate` concept, while an abstraction over `Terminate-Conversation` and `Terminate-Computer-Process`, still preserves the essential meaningful content of the two target concepts. While `Cause-New-State` does dominate `Infect-With-Cold` and `Induce-Idea` the essential content of these concepts lies in the concepts below the abstraction.

The relevance of this distinction becomes apparent when learning is considered. Consider the situation where an understander must determine the meaning of (7) by mapping from the metaphor underlying (6). In particular, consider the concretion step whereby the abstract common ancestor, `Cause-New-State`, is replaced by the most specific child that satisfies the constraints imposed by the input. In the *Kill* example, it was easy to uniquely concrete from `Terminate` to `Terminate-Computer-Process`

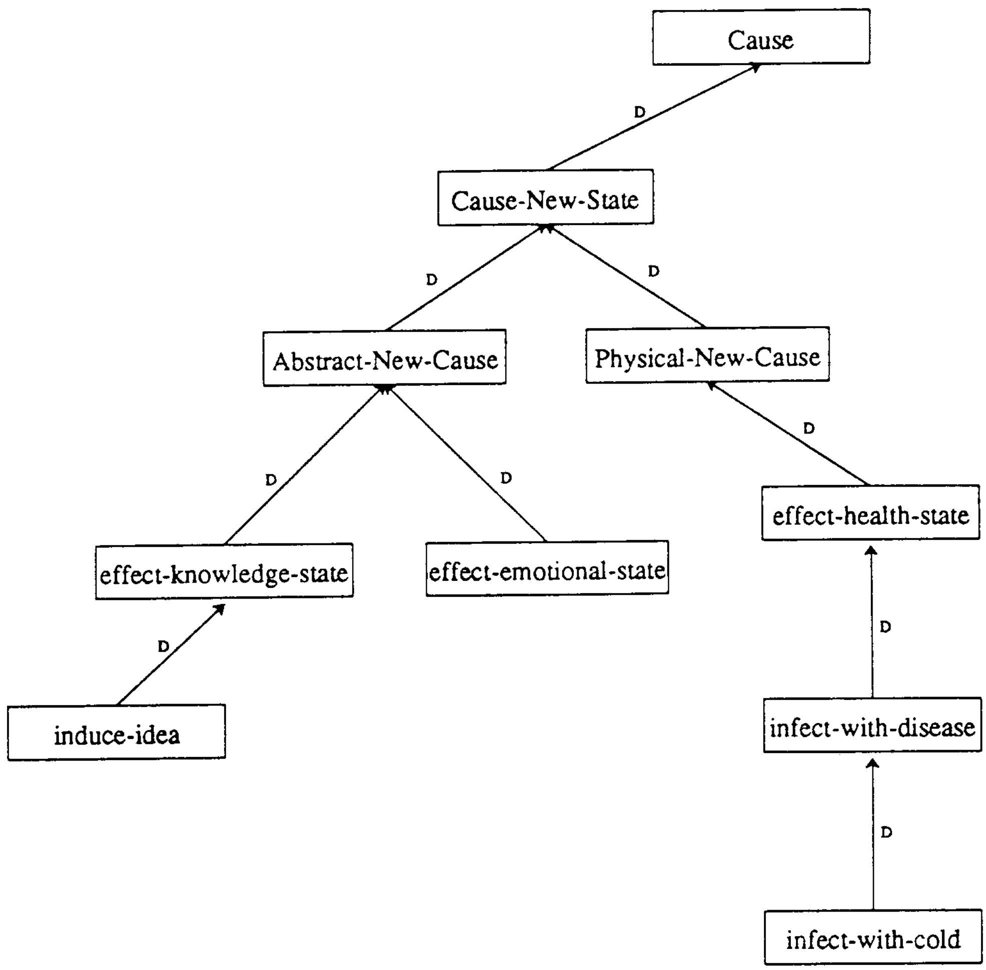

Figure 12: Cause-New-State Hierarchy

given the knowledge that the object being terminated was a `Computer-Process`.

The key to understanding why the concretion inference is possible in this example lies in the relationship between the common abstraction and the metaphor itself. It is not necessary for the abstraction to preserve detailed structure of the target concepts, as long as it preserves the structure that is the essence of the metaphor. Therefore, while the concept `Cause-New-State` loses most of the details of the individual target concepts, it does preserve enough of the structure referred to by the metaphor to allow a successful concretion to the correct specific target concept `Infect-With-Cold`. This is because the abstract concept `Cause-New-State` constrains the target to be an action that causes a person to enter a new state with respect to a cold. The only concept below `Cause-New-State` that preserves this structure is the concept `Infect-With-Cold`.

The key to the success of the approach is not in the overall degree of similarity that is preserved in the common abstraction. Instead, the important point is whether the common abstraction captures the essence of the metaphor. In these *giving* examples, it can be seen that the *giving* metaphors refer to a particular kind of causal concept that has to do with the creation of new states in an effected person. The distant relationship between the target concepts of *give a cold* and *give an idea* is sufficient to find this key concept.

8.5. Summary

This chapter has shown how the knowledge available in well understood similar metaphors can be used to understand new uses as they are encountered. A search technique has been introduced that exploits the abstraction hierarchy to find the intended target meaning of a new metaphor when given a similar known metaphor.

Chapter 9

Learning Extended Metaphors

9.1. Introduction

This chapter describes how a new metaphor can be understood by reference to a known conventional metaphor that is core-related to the new use. The Metaphor Preservation Principle, presented in Chapter 3, provides the basis for the strategy described in this chapter. The fundamental assertion of this chapter is that knowledge of the structure preservation predicted by the MPP allows extended metaphors to be incrementally acquired as new example metaphors are encountered.

Consider the metaphoric uses of the words *give*, *have*, and *get* in the following examples.

(1) Mary *has* a cold.
(2) John *gave* Mary a cold.
(3) Mary *got* her cold from John.

Consider the situation where a listener has specific knowledge of the metaphor underlying the use of *have* in (1). Specifically, it is known that there is a metaphorical use of *have* that has the concept `Infected-State` as the target meaning. Example (2) is then encountered. The task is to use the existing metaphorical use of *have* to search for the intended meaning of *give* in the target domain. This kind of learning is accomplished by a *Core-Extension* inference.

According to the *Metaphor Preservation Principle*, the concept filling the role of intended meaning must be one that is related to the known target, `Infected-State`, in such a way as to preserve the core-relation between `Giving` and `Having`. The key to learning the meaning of a new example is to use the core-relationship from the source domain to search for a concept in the target domain that will be related to the target of the existing metaphor in such a way as to preserve the source core-relationships. In the case of Example (2), the desired concept must be one that is related to the concept `Infected` in the same way that a `Giving` results in a `Having`. The concept `Infect-With-Disease`, which represents an action that causes a person to infect someone else with a disease, is chosen as the appropriate target. This choice is based on the preservation of the `Result` relation between `Having` and `Giving` in the relationship between `Infected` and `Infect-With-Disease`.

A core-extension inference is fundamentally a form of analogical reasoning. The existing core-related metaphor and the structure of the source and target domains form the basis for the analogy. Consider the following analogical formulation of the current example.

Having:Giving::Infected: ?

This formulation represents the situation that the MES finds itself in when it is presented with Example (2), and the `Have-Infection` is the candidate metaphor. The core-relationship in the source domain is the basis for the relationship between `Having` and `Giving` on the left side of the analogy. The pairing of `Having` and `Infected` in the analogy is a result of the source target structure of the known `Have-Infection` metaphor. The Metaphor Preservation Principle predicts that the intended target meaning of the new example will preserve the relation between `Having` and `Giving`, and thus complete the above analogy.

9.2. Core Extension Algorithm

The previous chapter discussed the details of the application step of the overall MES algorithm as it applied to similarity-related candidates. The rest of this chapter will discuss the application step in the case of core-related candidate metaphors. As in the previous chapter, the assumption is that the MES has been passed a primal representation for which there is no known coherent interpretation. In addition, the search and evaluation steps have chosen a known metaphor that is core-related to this new example. The rest of this chapter will show the details of how a core-related candidate metaphor can be used to determine the meaning of a new example.

The application of a core-related metaphor to a new example may be seen as a three step process. In preliminary processing, MES accepts a primal representation and finds a core-related candidate metaphor. In the next step, the exact nature of the core-relationship is determined. The details of this relationship form the basis for the remaining steps. The existing source core-relationship is applied to the target domain in an attempt to find an appropriate target concept. In the final step, the concept that results from the application of the core-relationship is concreted to the most specific known concept that can accept the input concepts.

Step 0: Accept a candidate metaphor and primal representation. The MES has been passed a primal representation for which there is no adequate conventional explanation. The MES, as a result of its first two steps, finds and selects a core-related candidate metaphor.

Step 1: Characterize the source core-relationship. This step determines the nature of the core-relationship between the input concepts and the candidate metaphor.

Step 2: Apply the source core-relationship to the target concepts. Using the input filler concepts and the source core-relation, choose the target concept that best preserves the source core-relationships in the target domain.

Step 3: Concrete the intended target concept found in Step 2 to the most specific concept that can accommodate the input concepts.

The following sections will give brief overviews of these steps. Detailed examples will be presented to illustrate the exact nature of the learning inferences for each of three possible core-relationships.

Step 1: Characterize. The core-relationship from the source domain consists of two parts. The first consists of the relation or relations that satisfy the Relation Condition. This relation will be referred to as the source core-relation. The source core-relation forms the basis of the hierarchical search in the target domain.

The second component consists of the containing relations between the input source and the core-related source concept. These containing relations consist of the equate links that indicate which input concepts play which roles in the core-related concept. The containing relations serve two purposes in the overall application of the target metaphor. The first purpose is as a check on the correctness of the concepts chosen as possible intended target meanings. The second use is to assign the input roles from the primal input to the appropriate roles in the core-related concepts.

Three kinds of relationships arise as a result of the core-relatedness conditions given above. Two of these arise from the asymmetry of the Containing Condition. In particular, the source input concept may contain or be contained by the source concept of the candidate metaphor. The third relationship arises from the Shared Intermediate Condition. These three relationships, in turn, give rise to three variations on the next step of the algorithm.

Step 2: Application. Application is the process by which the candidate metaphor is used to identify the intended target meaning of the new example. The core-relationship between the source concepts of the candidate metaphor and the source concepts underlying the new example is the key to the search in the target domain. The assumption is that the intended target concept will be the concept related to the target concept of the candidate metaphor that best preserves the core-relationships from the source domain.

The intended target concept is found by hierarchically matching the source relation against relations that are attached to the target concept of the candidate metaphor. A hierarchical match is a match that attempts to find a common ancestor between the two objects of the match. If they have a common ancestor it is returned; otherwise the match fails. The relation in the target domain that best matches the source core-relation will be attached to the intended target concept. The match that is considered best is the one that yields the most specific common ancestor.

Consider the situation where (2) is being understood by reference to the Infected-State-As-Possession metaphor underlying (1). The source core-relationship, in this case, is the `Giving-Result` relation, which indicates that a `Having` results from a `Giving`. The target concept of the known metaphor, `Infected-State`, has a number of relations attached to it. Of these relations, the relation `Infect-Result` best matches this source relation. These relations share the common ancestor relation, `Result`, which holds between causal actions and the states that result from them. The `Infect-Result` relation is, therefore, returned as the result of the matching step. The concept `Infect-With-Disease` which is linked to the known target concept `Infected-State`, by this relation, is chosen as the intended target concept.

The final step in the application phase is to use the containing structure as a check on the correctness of the target concept identified by relation matching. The concept found must satisfy the Containing conditions in the source. In other words, after it is shown that it hierarchically preserves the source core-relation, it then has to be shown that it has the appropriate containing structure.

The steps just outlined vary slightly based on the kind of core connection found in the source domain. In particular, three possible kinds of learning behavior arise from the three conditions for core-relatedness given in Chapter 3. These three possibilities affect how each of the above four steps is actually realized in an algorithm. The three kinds of learning behavior are called *Direct Extension*, *Realization*, and *Intermediate Extension*. These learning inferences are based on how the given new example is core-related to the candidate metaphor. Figure 1 summarizes the three basic kinds of core-extension inference. The details of each will be illustrated with examples in the following sections.

The first two inferences, Direct Extension and Realization, arise from the asymmetry of the Containing Condition for core-relatedness. This condition requires, in essence, that one of the concepts be completely contained by the other. In KODIAK terms, all the slots of one of the concepts must be equated to slots in the other concept. The concepts `Giving` and `Having` satisfy the Containing condition since the concept of `Giving` contains `Having` as a component. The two learning inferences arise based on whether the contained concept is the new example or the known metaphorical candidate. If the contained, or core concept, is the known metaphor, and the containing concept is the new example, then this is a Direct Extension inference. This is the case if the `Have-Infection` metaphor is understood and the new example involves the use of *give*. The *having* metaphor must be extended to cover the use of *give*. The new sense will necessarily involve concepts that are not contained within the existing metaphor and must be inferred from the target domain.

Direct Extension:

Extending a metaphor from a contained concept to a containing concept.

Example:

Extending *Having a cold* to *Giving a cold.*

Realization:

Extending a metaphor from a containing concept to a contained concept.

Example:

Extending *Giving a cold* to *Having a cold.*

Intermediate Extension:

Extending a metaphor from a concept to a core-related concept that shares an intermediate contained concept.

Example:

Extending *Giving a cold* to *Getting a cold* by going through *Having.*

Figure 1: Core-Extension Inferences

If the situation is reversed, and the known metaphor contains the definition of the new example, then this is a Realization inference. This would be the case when the known metaphor is the `Give-Infection` metaphor, and the new example involves the use of *have*. The term Realization is used because the new metaphor is in a sense already implicitly represented as a part of the containing concept. It is not so much necessary to extend the metaphor to new concepts as it is to explicitly move it to the appropriate place.

The final inference type is Intermediate Extension. This is the case when the known metaphor is core-related to the new use because they satisfy the Shared Intermediate Condition. Consider the case where the new example involves the use of *get*, and the known core-related metaphor is based on *give*. `Giving` and `Getting` are core-related by virtue of the fact that they share the concept `Having`. An Intermediate Extension inference must use the intermediate concept as the basis for inferring the new meaning. In practice, it will be shown that an intermediate extension inference consists first of a realization from the known metaphor to the shared core, followed by an extension from the core to the new use.

Step 3: Concrete the result of the previous step. It is possible that the core-related candidate metaphor exists at a higher level of abstraction than the input example. In this case, it may be necessary to concrete the concept found in the application step to a more specific concept. Like the concretion step in a Similarity Extension inference, this concretion is guided by the input filler concepts.

In the following sections, examples illustrating each of the three kinds of extension inference will be examined in detail. The assumption made in the following examples is that the first three steps of the MES algorithm have been completed. Therefore, a potentially useful core-related metaphor has already been identified. As was the case with the similarity learning described in Chapter 8, the emphasis will be on the knowledge that is available in a relevant candidate metaphor and how it can be utilized.

9.2.1. Direct Extension

Consider once again the following examples.

(4) Mary *has* a cold.
(5) John *gave* Mary a cold.

Again consider the situation where the metaphor underlying (4) is well understood, but an example like (5) using *give* has never been encountered. This section will describe the details of the Direct Extension inference that allows (5) to be understood through the use of the `Have-Infection` metaphor underlying (4). This involves a Direct Extension inference since `Having` and `Giving` are directly connected, and the concept involved in the existing metaphor, `Having`, is contained by the new example, `Giving`.

9.2.1.1. Step 0: Accept

Given (5) as an input example, the concept `Giving` is determined to be the source concept with `Cold` as a participant in the target concept. The search and evaluation phases have produced the `Have-Infection` metaphor as the best core-related candidate metaphor to work from. This metaphor specifies that the concept `Having` can metaphorically refer to the concept `Infected`, which represents the idea of a person being infected with some disease.

```
> (do-sentence)

Interpreting sentence:

John gave Mary a cold.

Interpreting primal input.

(A Giving34 (↑ Giving)
   (agent46 (↑ agent) (A John44 (↑ John)))
```

```
   (patient46 (↑ patient) (A Mary34 (↑ Mary)))
   (object34 (↑ object) (A Cold20 (↑ Cold))))

Concreting input relations.

  Concreting  object to given.
  Concreting  patient to givee.
  Concreting  agent to giver.

Interpreting concreted input.

(A Giving34 (↑ Giving)
   (giver34 (↑ giver) (A John44 (↑ John)))
   (givee34 (↑ givee) (A Mary34 (↑ Mary)))
   (given34 (↑ given) (A Cold20 (↑ Cold))))

Failed interpretation: Giving34 as Giving.

No valid interpretations. Attempting to extend existing metaphor.

===========================================================
Entering Metaphor Extension System
===========================================================

Searching for related known metaphors.

Metaphors found: Have-Infection Have-Idea Have-Permission Get-Grade
Selecting metaphor Have-Infection to extend from.

Attempting a core-related metaphorical extension.

Extending similar core-related metaphor Have-Infection
with target concept Cold-Inf-State.
```

9.2.1.2. Step 1: Characterization

This step determines the exact nature of the core-relationship between the candidate metaphor and the input example. In this example, the source concept of the candidate metaphor, `Having`, is related to the input concept, `Giving`, via the `giving-result` slot. Moreover, `Having` is completely contained by `Giving`, since all of its slots are equated to slots of `Giving`. This situation is indicative of a Direct Extension inference. It is necessary to extend the `Have-Infection` metaphor to cover the new use. Figure 2 shows the details of the core-relations between the source concepts `Giving` and `Having`, in addition to the `Have-Infection` metaphor connection to the target concepts.

The equate links that form the basis for the Containing Condition also provide bindings from the input filler concepts to the core-related concept through to the target of the candidate metaphor. In this example, the fillers of the `givee` and `given` roles, `Mary34` and `Cold20`, are bound through the containing relations to the core-related `haver` and `had` roles, and also to the `infected-person` and `infection-of-infected` target roles. These bindings will be carried through to the next step in order to assign the input fillers to the appropriate roles in the final target meaning.

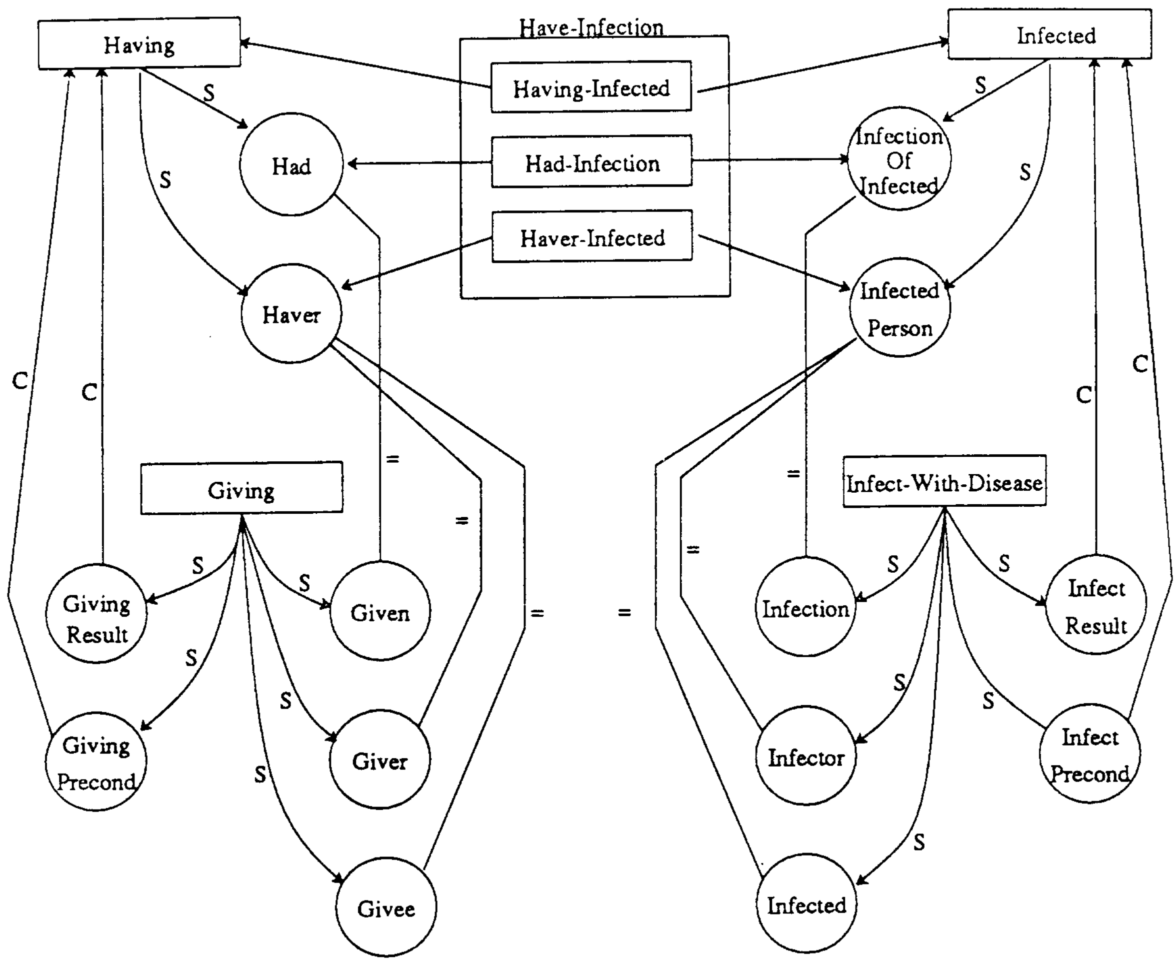

Figure 2: Giving and Having a Disease

9.2.1.3. Step 2: Application

The next step in the algorithm is the application step. This step hierarchically matches the source core-relation against relations attached to the target concept of the candidate metaphor. In this case, the relation underlying the `giving-result` slot is matched against the relations attached to the candidate target concept `Infected-State`. This is an attempt to find a concept related to `Infected` in a way that best preserves the core-relationship between `Giving` and `Having`.

Figure 3 shows some of the details of the candidate target concept, `Infected`. This figure shows this concept engaging in five relations. Two of the relations underlie the `infection-of-infected` and `infected-person` slots of the `Infected-State` concept. The other three relations represent connections between this concept and two other concepts that result in an `Infected-State`. The concept `Infect-With-Disease` represents the concept of an infected person causing another person to become infected. This concept is linked to `infected` via the `infect-result` slot. The other concept, `Become-Infected`, represents the event of a person coming down with a disease without specifying any cause.

When the `giving-result` slot is hierarchically matched against all the relations attached to `Infected` there are two successful matches. Remember that a successful match is one that yields a common ancestor. In this case, the `infect-result` and `become-infect-result` slots both have successful matches since they have the concept `result` as ancestors. This indicates that there are two known concepts that result in an `Infected` state.

The relation `infect-result` is chosen as the best match. The result of a successful match is the common ancestor found between the two concepts. In the case of multiple successful matches, the match that results in the most specific common ancestor is the one that is chosen. In this case, `infect-result` matches `giving-result` at the concept `action-result`, representing a state that results from a concept that is categorized as an action. The concept `become-infect-result` matches `giving-result` at the concept `result` that represents something that is the result of an event. This is a more abstract match and hence the `infect-result` is chosen.

Finding a concept that is in a core-relation to the candidate target that best preserves the relation from the source domain is only the first check on the possible target meaning. The intended meaning must also satisfy the structural constraints imposed by the Containing condition. In other words, the concept `Infect-With-Disease` must contain the concept `Infected` in the same way that the concept `Giving` contains the concept of `Having`. In cases where the constraints posed by core-relations alone cannot eliminate all the possible choices then the Containing conditions can further narrow the field. In this example, they merely serve as a further check on the single concept identified by the core-relations.

The Containing conditions are checked by making sure that the concept identified as the likely target meaning has slots that are equated to slots in the candidate target

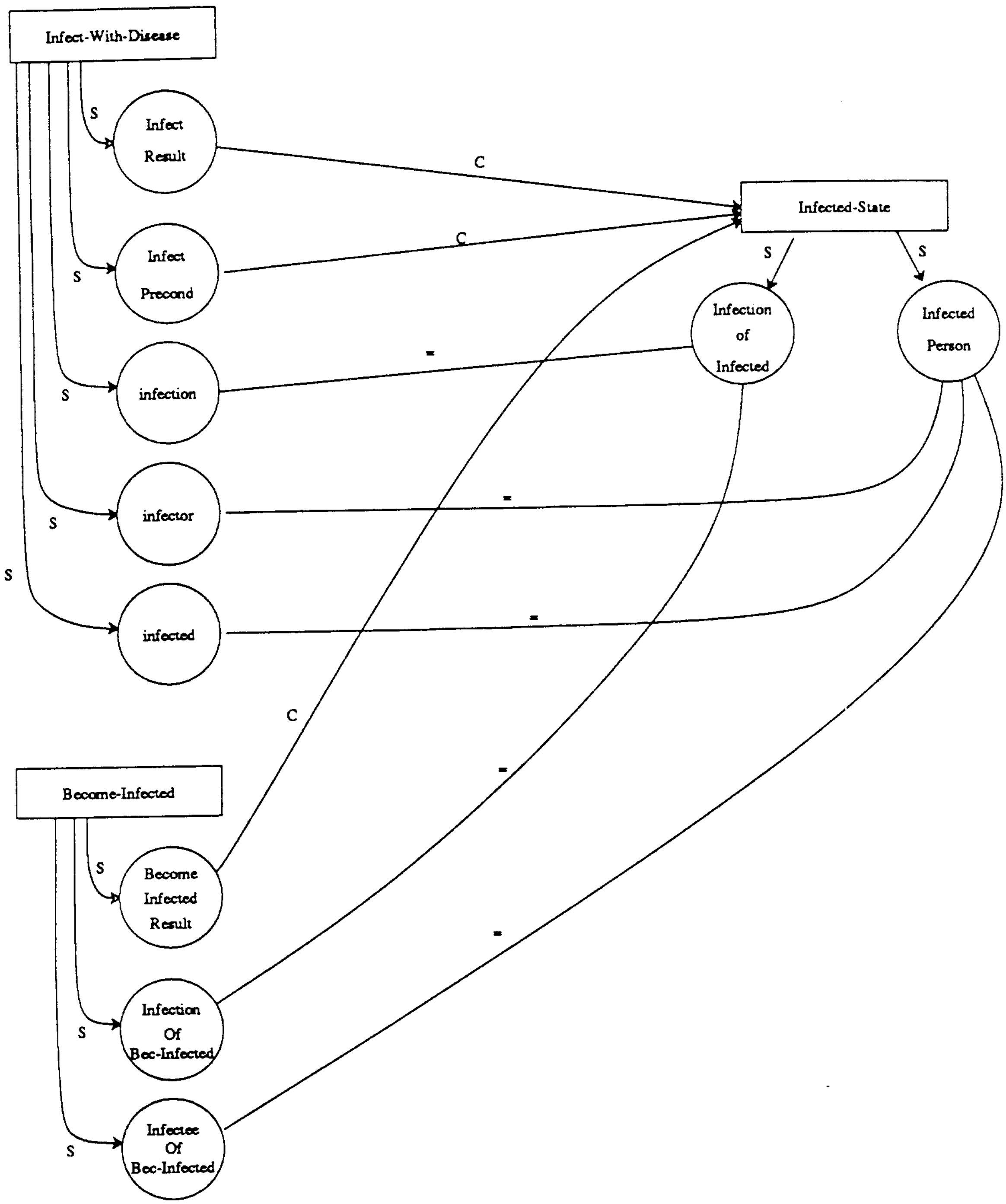

Figure 3: Infection

meaning in ways that reflect corresponding concepts in the source. In this example, the concept `Infect-With-Disease` should have one slot that is equated to the concept `infection-of-infected`. This reflects the equate relationship between the concepts `had` and `given` in the source. As shown in Figure 3, the `infection` slot satisfies this constraint. Finally, `Infect-With-Disease` must have another slot that is equated to the `infected-person` concept. This reflects the equate on the `givee` in the source. This is taken care of by the `infected` slot. The concept `Infect-With-Disease`, therefore, preserves all the appropriate core-relationships from the source domain.

```
This is a direct extension inference.

Applying source path:

Giving → give-result → Having

to target concept Infected yields target connection.

Infected → infect-result* → Infect-With-Disease


Applying source path yields target concept Infect-With-Disease

(A Infect-With-Disease5 (↑ Infect-With-Disease)
   (infectorll (↑ infector)  (A John44 (↑ John)))
   (infected13 (↑ infected)  (A Mary34 (↑ Mary)))
   (infection13 (↑ infection)  (A Cold20 (↑ Cold))))
```

9.2.1.4. Step 3: Concretion

The concept resulting from the application step may not be the most specific known concept that can accommodate the input concepts. Therefore, the concept resulting from the previous step is subject to a concretion inference in the same manner as in the Similarity Extension inference.

In this example, the general concept of infecting someone with a disease is replaced with the concept `Cold-Infect`. This represents the more specific concept of infecting someone with the common cold. This concretion is based upon the input filler concept `Cold20` in the role of `infection`. This more specific role more tightly accommodates the input concept `Cold20`. Since all the other roles are still valid in this concept, it is chosen as a more specific version of the intended target meaning.

```
Concretion yields:

(A Cold-Infect5 (↑ Cold-Infect)
   (cold-infectorll (↑ cold-infector)  (A John44 (↑ John)))
```

```
   (cold-victim13 (↑ cold-victim)        (A Mary34 (↑ Mary)))
   (infected-cold13 (↑ infected-cold)    (A Cold20 (↑ Cold))))

Mapping main source concept Giving to main target concept Cold-Infect.
 Mapping source role giver to target role cold-infector.
 Mapping source role givee to target role cold-victim.
 Mapping source role given to target role infected-cold.

(A Giving-Cold-Infect (↑ Metaphor-Schema)
   (given-infected-cold-map given → infected-cold)
   (givee-cold-victim-map givee → cold-victim)
   (giver-cold-infector-map giver → cold-infector)
   (giving-cold-infect-map Giving → Cold-Infect))

Final interpretation of input:

(A Cold-Infect5 (↑ Cold-Infect)
   (cold-infector11 (↑ cold-infector)    (A John44 (↑ John)))
   (cold-victim13 (↑ cold-victim)        (A Mary34 (↑ Mary)))
   (infected-cold13 (↑ infected-cold)    (A Cold20 (↑ Cold))))
```

9.2.2. Realization

Now consider the opposite situation to the previous example. Assume that the metaphor underlying the use of *give* in (5) is well-understood, while the use of *have* is new. In this case, the core-relationship from the source domain is one where the new use is wholly contained within the definition of a known metaphor. In this case, the definition of *have* is wholly contained in the definition of *give*. This type of inference is called Realization because, in a sense, the metaphor is already represented as a part of the known metaphor. A search in the target domain for the meaning of the new use is not required. Rather, what is needed is the explicit creation of a new metaphor sense at the level of the concept `Having`.

The application step for a Direct Extension inference required a search among the relations attached to the candidate target metaphor for one that hierarchically preserved the core-relation from the source. No such search is required for a Realization inference. This is because all of the concepts that will eventually take part in the new metaphor-sense already have a metaphorical correspondence implied by the known metaphor. This was not the case with the direct extension inference because the concept `Giving` did not already have a role in the known metaphor and hence had to be found.

```
> (do-sentence)

Interpreting sentence:

John has a cold.

Interpreting primal input.

(A Having5 (↑ Having)
   (agent47 (↑ agent) (A John45 (↑ John)))
   (patient47 (↑ patient) (A Cold21 (↑ Cold))))

Concreting input relations.

  Concreting  patient to had.
  Concreting  agent to haver.

Interpreting concreted input.

(A Having5 (↑ Having)
   (haver5 (↑ haver) (A John45 (↑ John)))
   (had5 (↑ had) (A Cold21 (↑ Cold))))


Failed interpretation: Having5 as Having.
Failed interpretation: Having5 as Have-Idea.
Failed interpretation: Having5 as Have-Permission.

No valid interpretations. Attempting to extend existing metaphor.

===========================================================
Entering Metaphor Extension System
===========================================================

Searching for related known metaphors.

Metaphors found: Give-Infection Have-Idea Have-Permission Get-Grade
Selecting metaphor Give-Infection to extend from.

Attempting a core-related metaphorical extension.

Extending similar core-related metaphor Give-Infection
with target concept Infect-With-Disease.
```

```
This is a realization inference.

Applying source path:

Having → give-result* → Giving

to target concept Infect-With-Disease yields target connection.

Infect-With-Disease → infect-result → Infected

Applying source path yields target concept Infected.

(A Infected2 (↑ Infected)
   (infected-person2 (↑ infected-person) (A John45 (↑ John)))
   (infection-of-infected2  (↑ infection-of-infected)
     (A Cold21 (↑ Cold))))

Concretion yields:

(A Cold-Inf-State2 (↑ Cold-Inf-State)
   (cold-inf-person2 (↑ cold-inf-person) (A John45 (↑ John)))
   (cold-inf-of2 (↑ cold-inf-of) (A Cold21 (↑ Cold))))


Mapping main source concept Having to main target concept Cold-Inf-State.
 Mapping source role haver to target role cold-inf-person.
 Mapping source role had to target role cold-inf-of.

(A Having-Cold-Inf-State (↑ Metaphor-Schema)
   (had-cold-inf-of-map had → cold-inf-of)
   (haver-cold-inf-person-map haver → cold-inf-person)
   (having-cold-inf-state-map Having → Cold-Inf-State))

Final interpretation of input:

(A Cold-Inf-State2 (↑ Cold-Inf-State)
   (cold-inf-person2 (↑ cold-inf-person) (A John45 (↑ John)))
   (cold-inf-of2 (↑ cold-inf-of) (A Cold21 (↑ Cold))))
```

In the current example, the core-relationship between `Having` and `Giving` specifies that the `giving-result` slot is constrained to be a `Having`. The `Give-Infection` metaphor specifies that the concept `infect-result` corresponds to the source concept `giving-result`. It follows, therefore, that the intended target meaning must be the concept that constrains the `infect-result` slot. A similar analysis follows for the assignment of the `haver` and `had` slots to the `infected-person` and `infection-of-infected` slots respectively.

9.2.3. Intermediate Extension

The final kind of learning inference arises from a core-relation based on the Shared Intermediate Condition. Again consider the situation where the `Give-Infection` metaphor is well-understood. A metaphorical use of *get* is then encountered as in (3). As described Chapter 3, the concept `Having` is shared by these two concepts, hence they are core-related. Assume again that the `Have-Infection` metaphor does not yet exist.

An intermediate extension inference is simply a combination of a Realization inference with a Direct Extension inference. The original metaphor is first extended via a Realization inference to the shared intermediate concept. The metaphor is then extended from the intermediate concept to the original example via a Direct Extension inference. In this example, the candidate metaphor is extended to the input concept `Getting` through the shared intermediate `Having`.

```
> (do-sentence)

Interpreting sentence:

John got a cold.

Interpreting primal input.

(A Getting7 (↑ Getting)
   (agent48 (↑ agent) (A John46 (↑ John)))
   (patient48 (↑ patient) (A Cold22 (↑ Cold))))

Concreting input relations.

  Concreting  patient to gotten.
  Concreting  agent to getter.

Interpreting concreted input.

(A Getting7 (↑ Getting)
   (getter7 (↑ getter) (A John46 (↑ John)))
   (gotten7 (↑ gotten) (A Cold22 (↑ Cold))))

Failed interpretation: Getting7 as Getting.
Failed interpretation: Getting7 as Get-Grade.
```

```
No valid interpretations. Attempting to extend existing metaphor.

============================================================
Entering Metaphor Extension System
============================================================

Searching for related known metaphors.

Metaphors found: Give-Infection Have-Idea Have-Permission Get-Grade
Selecting metaphor Give-Infection to extend from.

Attempting a core-related metaphorical extension.

Extending similar core-related metaphor Give-Infection
with target concept Infect-With-Disease.

This is an intermediate extension inference.


Applying source path:

 Getting → get-result → Having → give-result* → Giving

to target concept Cold-Infect yields target connection.

 Infect-With-Disease → infect-result → Infected →bec-infected-res*
  → Bec-Infected-With-Disease


Applying source path yields target concept Bec-Infected-With-Disease.

(A Bec-Infected-With-Disease5 (↑ Bec-Infected-With-Disease)
   (bec-inf-victim5 (↑ bec-inf-victim)    (A John46 (↑ John)))
   (bec-infected-inf5 (↑ bec-infected-inf) (A Cold22 (↑ Cold))))

Concretion yields:

(A Bec-Inf-W-Cold5 (↑ Bec-Inf-W-Cold)
   (bec-cold-victim5 (↑ bec-cold-victim)    (A John46 (↑ John)))
   (bec-infected-cold5 (↑ bec-infected-cold) (A Cold22 (↑ Cold))))


Mapping main source concept Getting to main target concept Bec-Inf-W-Cold.
 Mapping source role getter to target role bec-cold-victim.
 Mapping source role gotten to target role bec-infected-cold.

(A Getting-Bec-Inf-W-Cold (↑ Metaphor-Schema)
```

```
   (gotten-bec-infected-cold-map gotten → bec-infected-cold)
   (getter-bec-cold-victim-map getter → bec-cold-victim)
   (getting-bec-inf-w-cold-map Getting → Bec-Inf-W-Cold))


Final interpretation of input:

(A Bec-Inf-W-Cold5 (↑ Bec-Inf-W-Cold)
   (bec-cold-victim5 (↑ bec-cold-victim)    (A John46 (↑ John)))
   (bec-infected-cold5 (↑ bec-infected-cold) (A Cold22 (↑ Cold))))
```

In the current example, a Realization inference is performed from the concept `Giving` to the shared intermediate concept, `Having`, in exactly the same fashion as described in the previous section. The intended target meaning of *get* is then inferred from `Having` by a Direct Extension inference.

9.3. Relation to Analogy

As discussed in Chapter 2, there have been many previous approaches to metaphor that advocated an analogy-based solution to the problem. These approaches all attempted to compute a direct analogy between source and target domains, without reference to existing metaphorical structures. These approaches were all shown to be deficient in that the structure of the proposed analogical formulation was simply too unconstrained to be efficiently computable.

The approach described in this chapter uses the partial structure provided by existing core-related metaphors to provided a more tractable formulation. In particular, the previously understood metaphor provides the object correspondences that are necessary to any analogical approach. In addition to these correspondences, the core structure of the target domain of the known metaphor considerably reduces the number of concepts considered in the analogical matching phase.

9.4. Summary

This chapter has shown how the partial metaphorical structuring of a domain can be used to understand new metaphors that incrementally extend this structure. The strategy employed to accomplish this extension is based upon the Metaphor Preservation Principle. The use of this principle yields a formulation of the problem that corresponds to a simple analogical inference.

Chapter 10

Previous Literature Revisited

10.1. Introduction

This chapter demonstrates how MIDAS can handle some representative examples from the previous literature discussed in Chapter 2. The examples presented here will all be cases where MIDAS lacks an appropriate metaphor and proceeds to understand the new use by reference to an existing metaphor. The purpose of this chapter is not to simply show that my system can handle examples from previous systems. Rather, a reanalysis of these examples from a knowledge-based conventional metaphor perspective will be presented. Each example will be analyzed it terms of how it fits into a complex system of already understood metaphors. The following examples from the literature will be demonstrated.

(1) N is *at* zero.
(2) N *goes* from one to fifty.

(3) Britain *entered* the Common Market.
(4) Britain tried to *leave* the Common Market.

(5) Robbie's running *ate up* the distance.
(6) Inflation is *eating up* our savings.

(7) My car *drinks* gasoline.

(8) John *has* a solution.
(9) John *found* a solution.
(10) John *searched* for a solution.

This method of considering these examples will illustrate a number of points. First it will show how the MES can handle problems that these previous systems handled.

More importantly, however, the analysis will shed some light on why these previous systems worked as well as they did. These previous approaches, for the most part, analyzed these examples from the perspective of attempting to draw analogical inferences from the source to the target domains directly. This led to analyses that attempted to find or create similarities between the source and target concepts. We will see that some of the purported similarities between the source and target domains that these previous systems relied upon arise from the fact that the target domains were structured in terms of existing conventional metaphors. The power of these systems, therefore, arose not from their analogical reasoning capabilities, but rather from the implicit use of conventional metaphoric knowledge.

10.2. Hobbs

As discussed in Chapter 2, Hobbs (1979) makes a proposal that corresponds to the notion of the explicit representation of conventional metaphorical knowledge. He cites the following examples as evidence of the existence of a metaphor schema that identifies a variable having a value with the notion of location. He further goes on to say that (12) follows from (11) because *at* and *go* "are intricately woven together by their reference to a small set of common predicates". This is exactly the kind of phenomena captured by a core-relationship.

(11) N is *at* zero.
(12) N *goes* from one to fifty.

The location metaphor cited by Hobbs is a pervasive one in English. The use of location to indicate state is matched only by the use of possession to indicate state. The following examples will show how the MES can instantiate a specific metaphor from an abstract high level one to account for (11). This new specific metaphor will in turn be used as the basis for a core extension to cover the use of *go* in (12).

```
> (do-sentence)
Interpreting sentence:

N is at zero.

Interpreting primal input.

(A Is-At14 (↑ Is-At)
   (agent615 (↑ agent) (A N72 (↑ N)))
   (patient576 (↑ patient) (A Zero9 (↑ Zero))))

Concreting input relations.

  Concreting  patient to location-of.
  Concreting  agent to located-object.
```

```
Interpreting concreted input.

(A Is-At14 (↑ Is-At)
   (located-object11 (↑ located-object)
    (A N72 (↑ N)))
   (location-of11 (↑ location-of)
    (A Zero9 (↑ Zero))))

Failed interpretation: Is-At14 as Is-At.

Valid known metaphorical interpretation.

Applying conventional metaphor At-State.

(A At-State  (↑ Location-Metaphor Metaphor-Schema)
   (location-value location-of → state-value)
   (located-state-holder located-object → state-object)
   (is-at-state Is-At → State))
```

At this point, the system has identified the extremely abstract `At-State` metaphor as a valid interpretation of the input.

```
Mapping input concept Is-At14 to concept State77
  Mapping input role location-of11 with filler Zero9 to
       target role state-value102
  Mapping input role located-object11 with filler N72 to
       target role state-object90

Yielding interpretation:

(A State77 (↑ State)
   (state-value102 (↑ state-value)
    (A Zero9 (↑ Zero)))
   (state-object90 (↑ state-object)
    (A N72 (↑ N))))
```

The system then checks for valid concretions of the abstract concept `State77`. In this case, the system can successfully concrete the abstract concept down to the more specific concept `Variable-Value`.

```
Concretion yields:

(A Variable-Value35 (↑ Variable-Value)
   (variable-w-val36 (↑ variable-w-val)
    (A N72 (↑ N)))
```

```
   (value-of-variable36 (↑ value-of-variable)
    (A Zero9 (↑ Zero))))

Creating new metaphor:

 Mapping main source concept Is-At to main target concept Variable-Value.
  Mapping source role located-object to target role variable-w-val.
  Mapping source role location-of to target role value-of-variable.

(A Is-At-Variable-Value (↑ At-State)
   (location-of-value-of-variable-map location-of → value-of-variable)
   (located-object-variable-w-val-map located-object → variable-w-val)
   (is-at-variable-value-map Is-At → Variable-Value))
```

This instantiation of a new metaphor is a behavior of the system that has not been previously demonstrated. The MES was not called because a valid known metaphor was found and applied. The system, however, decided to instantiate a more specific metaphor. This behavior will be described more fully below.

```
Final interpretation of input:

(A Variable-Value35 (↑ Variable-Value)
   (variable-w-val36 (↑ variable-w-val)
    (A N72 (↑ N)))
   (value-of-variable36 (↑ value-of-variable)
    (A Zero9 (↑ Zero))))
```

In this example, the system found and applied a known metaphor to obtain the target concept. This concept was then concreted to the intended meaning. In previous cases, this process did not result in the creation of a new more specific metaphor. In those cases where a concretion is made from the target concept of the metaphor to a more specific concept, the system keeps track of how difficult the concretion was to make. This difficulty is measured in the number of concepts that the concretion process had to consider before an appropriate meaning was found. In those cases where this difficulty ranking is above a certain threshold, a new more specific metaphor is created. In future processing the new metaphor can be applied directly without having to perform a costly concretion.

In this case, the system found that the concretion from the abstract concept `State` to the concept `Variable-Value` was difficult enough to warrant the creation of a new more specific metaphor representing the location for variable-value metaphor.

Once the new specific core metaphor has been generated from the abstract metaphor, core extension inferences are used to elaborate the core based on new input examples. The previous example instantiated the new metaphor `Is-At-Variable-Value`. The following example elaborates it to cover the use of *go* in (12).

```
> (do-sentence)

Interpreting sentence:

N goes from one to fifty.

Interpreting primal input.

(A Moves-To2 (↑ Moves-To)
   (agent66 (↑ agent) (A N14 (↑ N)))
   (to2 (↑ to) (A Fifty12 (↑ Fifty)))
   (from2 (↑ from) (A One2 (↑ One))))

Concreting input relations.

  Concreting  from to moved-from.
  Concreting  to to moved-to.
  Concreting  agent to moved.

Interpreting concreted input.

(A Moves-To2 (↑ Moves-To)
   (moved2 (↑ moved) (A N14 (↑ N)))
   (moved-to2 (↑ moved-to)
    (A Fifty12 (↑ Fifty)))
   (moved-from2 (↑ moved-from)
    (A One2 (↑ One))))


Failed interpretation: Moves-To2 as Moves-To.

No valid interpretations. Attempting to extend existing metaphor.

===========================================================
Entering Metaphor Extension System
===========================================================


Searching for related known metaphors.

Metaphors found: Is-At-Variable-Value At-State

Selecting metaphor Is-At-Variable-Value to extend from.
```

```
(A Is-At-Variable-Value (↑ At-State)
   (location-of-value-of-variable-map location-of → value-of-variable)
   (located-object-variable-w-val-map located-object → variable-w-val)
   (is-at-variable-value-map Is-At → Variable-Value))
```

The search phase identifies two metaphors as being potentially useful. One is the original `At-State` metaphor the other is the newly created `Is-At-Variable-Value` metaphor. The evaluation phase chooses the metaphor that is closest to the input concepts in conceptual distance. The roles in the `Is-At-Variable-Value` metaphor are more specific, and hence are closer to the input roles of a variable and a number than the more abstract roles in the `Is-At-Variable-Value` metaphor.

```
Attempting a core-related metaphorical extension.

Extending similar core-related metaphor Is-At-Variable-Value
with target concept Variable-Value.

This is a direct extension inference.

Applying source path:

Moves-To → move-to-res → Is-At

to target concept Variable-Value yields target connection.

Variable-Value → ch-var-val-res* → Change-Variable-Value


Applying source path yields target concept Change-Variable-Value.

(A Change-Variable-Value1
   (↑ Change-Variable-Value)
   (variable-changed1 (↑ variable-changed)
    (A N14 (↑ N)))
   (final-variable-value1
    (↑ final-variable-value)
    (A Fifty12 (↑ Fifty))))

Creating new metaphor:

 Mapping main source concept Moves-To to main target concept
Change-Variable-Value.
  Mapping source role moved to target role variable-changed.
  Mapping source role moved-to to target role final-variable-value.

(A Moves-To-Change-Variable-Value  (↑ Metaphor-Schema)
```

```
(moved-to-final-variable-value-map moved-to → final-variable-value)
(moved-variable-changed-map moved → variable-changed)
(moves-to-change-variable-value-map Moves-To → Change-Variable-Value))
```

A new metaphor is now created to represent the conceptual use of changing location to mean a change of a variable's value.

```
Final interpretation of input:

(A Change-Variable-Value1
   (↑ Change-Variable-Value)
   (variable-changed1 (↑ variable-changed)
    (A N14 (↑ N)))
   (final-variable-value1
    (↑ final-variable-value)
    (A Fifty12 (↑ Fifty))))
```

10.3. Wilks

As described in Chapter 2, Wilks (1978) proposed a system that could infer the meaning of the metaphor in (13), when given a complete episodic representation that includes the intended target concept in the representation.

(13) Britain *entered* the Common Market.
(14) Britain tried to *leave* the Common Market.

Wilks points out that even after such a system has understood (13), Example (14) still can not be handled. This is because there can be no representation of the event of Britain leaving the Common Market in the context since it has never occurred. The obvious problem with this approach is that it ignores the long term semantic connections in the source domain. In particular, the core relationships among concepts in the source domain are not used. In this case, the relationships of interest are those involving *entering*, *leaving*, and being *in* enclosures or environments. The following example shows how the MES can make use of this kind of knowledge.

```
> (do-sentence)
Interpreting sentence:

Britain tried to leave the common market.

Interpreting primal input.
```

```
(A Leaving22 (↑ Leaving)
   (agent607 (↑ agent)
    (A Britain47 (↑ Britain)))
   (patient570 (↑ patient)
    (A Common-Market23 (↑ Common-Market))))

Concreting input relations.

  Concreting  patient to left.
  Concreting  agent to leaver.

Interpreting concreted input.

(A Leaving22 (↑ Leaving)
   (leaver22 (↑ leaver)
    (A Britain47 (↑ Britain)))
   (left22 (↑ left)
    (A Common-Market23 (↑ Common-Market))))

Failed interpretation: Leaving22 as Leaving.

No valid interpretations. Attempting to extend existing metaphor.

===========================================================
Entering Metaphor Extension System
===========================================================


Searching for related known metaphors.

Metaphors found: Enter-Association Enter-Lisp

Selecting metaphor Enter-Association to extend from.

(A Enter-Association (↑ Enter-Metaphor Metaphor-Schema)
   (association-enterer enterer → associator)
   (entered-association entered → association-of-associate)
   (enter-associate Entering → Associate-Action))

Attempting a core-related metaphorical extension.
```

The system finds and selects a known core-related metaphor that represents the idea that the action of associating with an association can be viewed as entering the association.

```
Extending similar core-related metaphor Enter-Association
with target concept Associate-Action.

This is an intermediate extension inference.

Applying source path:

 Leaving → leave-pre → Enclosed-State → enter-res* → Entering

to target concept Associate-Action yields target connection.

 Associate-Action → associate-result → Associated-State
→ disassociate-pre* → Disassociate-Action

Applying source path yields target concept Disassociate-Action.
```

The system has applied the long term semantic connections between entering and leaving and applied them to the target domain to yield the following concept.

```
(A Disassociate-Action22 (↑ Disassociate-Action)
   (disassociator22 (↑ disassociator)
    (A Britain47 (↑ Britain)))
   (disassociated22 (↑ disassociated)
    (A Common-Market23 (↑ Common-Market))))

Creating new metaphor:

 Mapping main source concept Leaving to main target concept
Disassociate-Action.
  Mapping source role leaver to target role disassociator.
  Mapping source role left to target role disassociated.

(A Leaving-Disassociate-Action (↑ Metaphor-Schema)
   (left-disassociated-map left → disassociated)
   (leaver-disassociator-map leaver → disassociator)
   (leaving-disassociate-action-map Leaving → Disassociate-Action))

Final interpretation of input:

(A Trying24 (↑ Trying)
   (tried20 (↑ tried)
    (A Disassociate-Action22
       (↑ Disassociate-Action)
       (disassociator22 (↑ disassociator)
```

```
    (A Britain47 (↑ Britain)))
      (disassociated22 (↑ disassociated)
    (A Common-Market23 (↑ Common-Market)))))
  (agent606 (↑ agent)
   (A Britain47 (↑ Britain))))
```

10.4. DeJong and Waltz

DeJong and Waltz (1983) present a model of metaphor understanding based purely on analogical matching of source and target concepts directly. Consider the following example.

(15) Robbie's running *ate up* the distance.

The proposed processing for this example suggests an analogical match between eating and running that focusses on the fact that eating reduces the available amount of food, and running reduces a distance. The problem with this approach and others like it is the necessary ability to focus on this notion of reduction as the relevant concept among all the various concepts underlying *running* and *eating*. In MIDAS, this focus is obtained by using previously understood metaphors as a guide. This use of eat-up to mean reduce an amount has already been demonstrated by (16), discussed in Chapter 5.

(16) Inflation is eating up our savings.

Instead of attempting to match eat against running to obtain reduce, the system uses this previous use of eat-up to mean reduce an amount.

```
> (do-sentence)
Interpreting sentence:

Robbie 's running ate up the distance.

Interpreting primal input.

(A Running26 (↑ Running)
   (agent609 (↑ agent) (A Robbie29 (↑ Robbie))))
```

In this example, the system determines that there are two primal input concepts that are judged to require further interpretation. The first is the *running* in the subject of the sentence. This is found to have an appropriate literal interpretation.

```
Concreting input relations.

  Concreting  agent to runner.

Interpreting concreted input.

(A Running26 (↑ Running)
   (runner26 (↑ runner)
    (A Robbie29 (↑ Robbie))))

Valid literal interpretation.

(A Running26 (↑ Running)
   (runner26 (↑ runner)
    (A Robbie29 (↑ Robbie))))

Final interpretation:

(A Running26 (↑ Running)
   (runner26 (↑ runner)
    (A Robbie29 (↑ Robbie))))
```

The next clause contains the concept eating-up underlying *ate up*. This also requires further interpretation.

```
Interpreting primal input.

(A Eating-Up62 (↑ Eating-Up)
   (agent608 (↑ agent)
    (A Running26 (↑ Running)
       (runner26 (↑ runner)
      (A Robbie29 (↑ Robbie)))))
   (patient571 (↑ patient)
    (A Distance31 (↑ Distance))))

Concreting input relations.

  Concreting  agent to eater-of-eating-up.
  Concreting  patient to eaten-up.

Interpreting concreted input.

(A Eating-Up62 (↑ Eating-Up)
   (eater-of-eating-up68
```

```
    (↑ eater-of-eating-up)
    (A Running26 (↑ Running)
       (runner26 (↑ runner)
        (A Robbie29 (↑ Robbie)))))
   (eaten-up40 (↑ eaten-up)
    (A Distance31 (↑ Distance))))

Failed interpretation: Eating-Up62 as Eating-Up.
Failed interpretation: Eating-Up62 as Eat-Up-Reduce-Money.

No valid interpretations. Attempting to extend existing metaphor.

===========================================================
Entering Metaphor Extension System
===========================================================


Searching for related known metaphors.

Metaphors found: Eat-Up-Reduce-Money

Selecting metaphor Eat-Up-Reduce-Money to extend from.

(A Eat-Up-Reduce-Money  (↑ Eating-Metaphor Metaphor-Schema)
   (eaten-reduced eaten-up → money-reduced)
   (eater-reducer eater-of-eating-up → money-reducer)
   (eatup-reduce Eating-Up → Money-Loss))
```

The system has found and will now attempt to apply the previously understood metaphorical use of eat-up.

```
Attempting a similarity extension inference.

Extending similar metaphor Eat-Up-Reduce-Money with target concept Money-Loss.

Abstracting Money-Loss to ancestor concept Reduce-Amount producing
abstract target meaning:

(A Reduce-Amount47 (↑ Reduce-Amount)
   (reduced15 (↑ reduced)
    (A Distance31 (↑ Distance)))
   (reducer45 (↑ reducer)
    (A Running26 (↑ Running)
       (runner26 (↑ runner)
        (A Robbie29 (↑ Robbie))))))
```

```
Concreting target concept Reduce-Amount to Reduce-Distance producing
concreted meaning:

(A Reduce-Distance23 (↑ Reduce-Distance)
   (distance-reducer24 (↑ distance-reducer)
    (A Running26 (↑ Running)
       (runner26 (↑ runner)
     (A Robbie29 (↑ Robbie)))))
   (distance-reduced23 (↑ distance-reduced)
    (A Distance31 (↑ Distance))))

Creating new metaphor:

 Mapping main source concept Eating-Up to main target concept
  Reduce-Distance.
  Mapping source role eater-of-eating-up to target role
  distance-reducer.
  Mapping source role eaten-up to target role distance-reduced.

(A Eating-Up-Reduce-Distance  (↑ Eating-Metaphor)
   (eaten-up-distance-reduced-map eaten-up → distance-reduced)
   (eater-of-eating-up-distance-reducer-map eater-of-eating-up
    → distance-reducer)
   (eating-up-reduce-distance-map Eating-Up → Reduce-Distance))
```

The final interpretation of the sentence reflects the fact that the `running` is responsible for a reduction in distance.

```
Final interpretation of input:

(A Reduce-Distance23 (↑ Reduce-Distance)
   (distance-reducer24 (↑ distance-reducer)
    (A Running26 (↑ Running)
       (runner26 (↑ runner)
     (A Robbie29 (↑ Robbie)))))
   (distance-reduced23 (↑ distance-reduced)
    (A Distance31 (↑ Distance))))
```

10.5. Fass

Consider the following example from Fass (1987).

(17) My car *drinks* gasoline.

Fass makes extensive use of hierarchies to compute metaphorical analogies from the source to target domains. In interpreting this example, Fass uses the hierarchical relationships between the constraints on the literal meaning of the concept `Drinking` and the input examples to obtain an abstraction that partially accounts for this use. The relationships focussed on are that drinking prefers that the thing drunk be a potable liquid, which is hierarchically related to gasoline at the concept liquid.

Fass's approach is basically the same as performing the abstraction stage of a similarity extension inference on the source domain directly. In MIDAS, the abstraction is applied to the target concept of a known metaphor that is presumably much closer conceptually to the intended meaning than the source concept is. The other major difference is that Fass makes no attempt to concrete the abstract concept to an appropriately more specific concept. In this example, the concept returned is that a car `uses` gasoline. Where `use` is an abstract predicate covering any kind of using.

The meaning of *drink* in this example seems to focus more on the notion of a car consuming or using up its gasoline rather than the fact that it uses it to function. In the following example, the MES uses the previously understood *eating-up* metaphors that have as target meanings concepts having to do with reducing amounts. The system finds these metaphors by noticing the close core relationship between *drinking* and *eating-up*.

```
> (do-sentence)
Interpreting sentence:

My car drinks gasoline.

Interpreting primal input.

(A Drinking22 (↑ Drinking)
   (agent610 (↑ agent) (A Car28 (↑ Car)))
   (patient572 (↑ patient)
    (A Gasoline22 (↑ Gasoline))))

Concreting input relations.

  Concreting  patient to drunk.
  Concreting  agent to drinker.

Interpreting concreted input.
```

```
(A Drinking22 (↑ Drinking)
   (drinker18 (↑ drinker) (A Car28 (↑ Car)))
   (drunk18 (↑ drunk)
    (A Gasoline22 (↑ Gasoline))))
```

Failed interpretation: Drinking22 as Drinking.

No valid interpretations. Attempting to extend existing metaphor.

```
==========================================================
Entering Metaphor Extension System
==========================================================
```

Searching for related known metaphors.

Metaphors found: Eat-Up-Reduce-Money Eating-Up-Reduce-Distance

Selecting metaphor Eat-Up-Reduce-Money to extend from.

```
(A Eat-Up-Reduce-Money  (↑ Eating-Metaphor Metaphor-Schema)
   (eaten-reduced eaten-up → money-reduced)
   (eater-reducer eater-of-eating-up → money-reducer)
   (eatup-reduce Eating-Up → Money-Loss))
```

Attempting a core-related metaphorical extension.

Extending similar core-related metaphor Eat-Up-Reduce-Money
with target concept Money-Loss.

This is an intermediate extension inference.

Applying source path:

Drinking → drinking-result → Ingested-State → eaten-up-result*
→ Eating-Up

to target concept Money-Loss yields target connection.

Money-Loss → money-loss-result → Money-Loss-State → money-loss-result*
→ Money-Loss

Applying source path yields target concept Money-Loss.

Abstracting Money-Loss to ancestor concept Reduce-Amount producing
abstract target meaning:

```
(A Reduce-Amount48 (↑ Reduce-Amount)
   (reducer16 (↑ reducer)
    (A Car28 (↑ Car)))
   (reduced16 (↑ reduced)
    (A Gasoline22 (↑ Gasoline))))

Concreting Reduce-Amount48 to concept Reduce-Amount.

Yielding concept:

(A Reduce-Amount48 (↑ Reduce-Amount)
   (reducer16 (↑ reducer)
    (A Car28 (↑ Car)))
   (reduced16 (↑ reduced)
    (A Gasoline22 (↑ Gasoline))))

Creating new metaphor:

 Mapping main source concept Drinking to main target concept Reduce-Amount.
  Mapping source role drinker to target role reducer.
  Mapping source role drunk to target role reduced.

(A Drinking-Reduce-Amount (↑ Metaphor-Schema)
   (drunk-reduced-map drunk → reduced)
   (drinker-reducer-map drinker → reducer)
   (drinking-reduce-amount-map Drinking → Reduce-Amount))

Final interpretation of input:

(A Reduce-Amount48 (↑ Reduce-Amount)
   (reducer16 (↑ reducer)
    (A Car28 (↑ Car)))
   (reduced16 (↑ reduced)
    (A Gasoline22 (↑ Gasoline))))
```

10.6. Russell

Russell (1976) presents a series of examples that use Conceptual Dependency as the underlying representation. The approach taken is another heuristic matching algorithm that directly matches source and target domains. There is, however, an interesting facet to Russell's examples. The systems success is derived to a large extent from the matcher's ability to match the conceptual primitives PTRANS and MTRANS. These are represented as similar concepts. A PTRANS is a kind of transfer of a physical object.

An MTRANS is taken to be a kind of transfer where thoughts are the objects transferred and the locations of the transfers are taken to be minds. Given this structuring there is a high degree of match between the source and target domains of her examples.

As noted in Chapter 2, the CD primitive MTRANS is based on an underlying set of conventional metaphors. The metaphors that an idea is an object, and that the transfer and possession of these objects refers to knowing and communicating ideas, are codified directly into the primitive MTRANS. This reflects the fact that this is a highly embedded core metaphor in English. Consider the following metaphors from Russell.

(18) John *has* a solution.
(19) John *found* a solution.
(20) John *searched* for a solution.

Russell matches the various PTRANS and POSS concepts underlying the uses of *has*, *search*, and *find* against MTRANS and MPOSS concepts to yield the correct meanings. In the following examples, the MES uses an explicit representation of the conventional metaphor that possession is knowing, underlying (18), to interpret (19). The metaphor learned in that example is then further extended to cover (20).

```
> (do-sentence)
Interpreting sentence:

John found a solution.

Interpreting primal input.

(A Finding9 (↑ Finding)
   (agent612 (↑ agent) (A John149 (↑ John)))
   (patient573 (↑ patient)
    (A Solution20 (↑ Solution))))

Concreting input relations.

  Concreting  patient to found.
  Concreting  agent to finder.

Interpreting concreted input.

(A Finding9 (↑ Finding)
   (finder9 (↑ finder) (A John149 (↑ John)))
   (found9 (↑ found)
    (A Solution20 (↑ Solution))))

Failed interpretation: Finding9 as Finding.
```

```
No valid interpretations. Attempting to extend existing metaphor.

=============================================================
Entering Metaphor Extension System
=============================================================


Searching for related known metaphors.

Metaphors found: Have-Idea Have-Permission
Needing-Required-Permission
Giving-Permit-Action Have-Cold Have-State Get-Grade Give-Flu

Selecting metaphor Have-Idea to extend from.

(A Have-Idea (↑ Have-State Metaphor-Schema)
   (had-known had → known)
   (haver-knower haver → knower)
   (have-idea-map Having → Knowing))

Attempting a core-related metaphorical extension.

Extending similar core-related metaphor Have-Idea
with target concept Knowing.

This is a direct extension inference.

Applying source path:

Finding → find-result → Having

to target concept Knowing yields target connection.

Knowing → conclude-result* → Concluding


Applying source path yields target concept Concluding.

(A Concluding12 (↑ Concluding)
   (concluder11 (↑ concluder)
    (A John149 (↑ John)))
   (concluded7 (↑ concluded)
    (A Solution20 (↑ Solution))))

Creating new metaphor:

 Mapping main source concept Finding to main target concept Concluding.
```

```
  Mapping source role finder to target role concluder.
  Mapping source role found to target role concluded.

(A Finding-Concluding (↑ Metaphor-Schema)
   (found-concluded-map found → concluded)
   (finder-concluder-map finder → concluder)
   (finding-concluding-map Finding → Concluding))
```

A new metaphor is now created to represent the idea that coming to a conclusion can be viewed as finding an idea.

```
Final interpretation of input:

(A Concluding12 (↑ Concluding)
   (concluder11 (↑ concluder)
    (A John149 (↑ John)))
   (concluded7 (↑ concluded)
    (A Solution20 (↑ Solution))))
```

In the previous example, the system applied the core related `Have-Idea` metaphor to the metaphorical use of `finding`. Once this has been understood it creates a new metaphor representing this use. This new metaphor can now be extended to cover the following use of *search* based on the core relationship between searching and finding.

```
> (do-sentence)
Interpreting sentence:

John searched for a solution.

Interpreting primal input.

(A Searching7 (↑ Searching)
   (agent78 (↑ agent) (A John14 (↑ John)))
   (for7 (↑ for) (A Solution13 (↑ Solution))))

Concreting input relations.

  Concreting  for to searched-for.
  Concreting  agent to searcher.

Interpreting concreted input.

(A Searching7 (↑ Searching)
```

```
  (searcher7 (↑ searcher) (A John14 (↑ John)))
  (searched-for7 (↑ searched-for)
   (A Solution13 (↑ Solution))))
```

```
Failed interpretation: Searching7 as Searching.

No valid interpretations. Attempting to extend existing metaphor.

===========================================================
Entering Metaphor Extension System
===========================================================


Searching for related known metaphors.

Metaphors found: Finding-Concluding Have-Idea Searching-Investigating
 Have-Permission  Needing-Required-Permission Giving-Permit-Action
 Have-Cold Have-State Get-Grade Give-Flu


Selecting metaphor Finding-Concluding to extend from.

(A Finding-Concluding (↑ Metaphor-Schema)
   (found-concluded-map found → concluded)
   (finder-concluder-map finder → concluder)
   (finding-concluding-map Finding → Concluding))
```

The system finds and decides to apply the newly created finding metaphor.

```
Attempting a core-related metaphorical extension.

Extending similar core-related metaphor Finding-Concluding
with target concept Concluding.

This is a direct extension inference.

Applying source path:

Searching → intended-result → Finding

to target concept Concluding yields target connection.

Concluding → investigated-result* → Investigating
```

```
Applying source path yields target concept Investigating.

(A Investigating4 (↑ Investigating)
   (investigator5 (↑ investigator)
    (A John14 (↑ John)))
   (investigated4 (↑ investigated)
    (A Solution13 (↑ Solution)))
   (investigated-for5 (↑ investigated-for)
    (A Solution13 ^^^)))

 Mapping main source concept Searching to main target concept Investigating.
  Mapping source role searcher to target role investigator.
  Mapping source role searched-for to target role investigated.

(A Searching-Investigating (↑ Metaphor-Schema)
   (searched-for-investigated-map searched-for → investigated)
   (searcher-investigator-map searcher → investigator)
   (searching-investigating-map Searching → Investigating))


Final interpretation of input:

(A Investigating4 (↑ Investigating)
   (investigator5 (↑ investigator)
    (A John14 (↑ John)))
   (investigated4 (↑ investigated)
    (A Solution13 (↑ Solution)))
   (investigated-for5 (↑ investigated-for)
    (A Solution13 ^^^)))
```

10.7. Summary

A reanalysis of some examples from the previous literature shows that they can effectively be handled by the MIDAS approach. In addition, the reanalysis of these previous examples in terms of conventional metaphor shed some light on the workings of these previous systems. In particular, the seemingly ad-hoc construction of similar source and target domains in these approaches typically arises from the researcher's incorporation of conventional metaphors directly into the representation of various target domains.

Chapter 11

Conclusions

11.1. Introduction

The MIDAS approach to metaphor developed as a reaction against previous knowledge-deficient approaches. Two factors characterize these approaches: powerful special purpose analogy programs, and little or no explicit knowledge about the metaphors in the language. In contrast, MIDAS uses large amounts of specific knowledge about the metaphors in the language, and relies upon processes that are fundamentally the same as those already needed to interpret non-metaphorical knowledge. This approach arises from the simple belief that metaphor is a normal conventional part of language.

MIDAS has demonstrated the effectiveness of this knowledge-based approach. In particular, it has achieved the following three results: it is possible to capture systematic knowledge about metaphor using straightforward knowledge representation techniques, this knowledge can be efficiently applied to interpret metaphoric language, and, finally, this knowledge can be systematically extended as new metaphors are encountered. The following sections will briefly review these results and then discuss some problems.

11.2. Representation

A detailed analysis of what needs to be represented necessarily precedes the task of constructing a knowledge base. This analysis should reveal the phenomena that needs to be captured and suggest certain requirements for how these phenomena should be captured. The analysis of conventional metaphor, given in Chapter 3, resulted in the following requirements for the representation of metaphoric language.

> **Fundamental Representational Requirement:** Conventional metaphors must be explicitly represented as coherent sets of associations between source and target concepts.

Extended Metaphor Requirement: The representation of conventional metaphors must capture the fundamental phenomenon that a sharing of important core concepts underlies the perceived relationships among many separate metaphors.

Sharing Requirement: The representation of the component associations of metaphors must facilitate the hierarchical sharing of these component parts among distinct metaphors.

Similarity Requirement: The representation of conventional metaphors must capture the similarity relationships among metaphors.

Chapter 4 demonstrated how these requirements were met through the use of the KODIAK representation language. The key to the use of KODIAK was the decision to elevate metaphors and their component associations to the level of full-fledged KODIAK concepts. The combination of the structured association and inheritance mechanisms of KODIAK successfully satisfies these representational requirements.

11.3. Interpretation

The fundamental motivation behind the explicit representation of this metaphoric knowledge was to capture the intuition that metaphors are a normal and conventional part of language. The normal processing of metaphoric language should proceed through the application of this specific knowledge. Moreover, this processing should as much as possible reflect the belief that metaphor is an ordinary aspect of language. The mechanisms proposed should not be fundamentally different from those used to understand non-metaphorical language.

MIDAS has successfully achieved these goals. The concretion and metaphoric unviewing inferences that select an appropriate interpretation are fundamentally the same kind of constraint checking mechanisms. These mechanisms choose among the plausible conventional interpretations by seeing how well each interpretation meets the constraints imposed by the utterance. This constraint driven choice of interpretation is fundamentally the same for both metaphoric and non-metaphoric language. The selection or rejection of a metaphoric interpretation is based solely upon how well it meets these constraints. It is in no way dependent upon the well-formedness of the literal reading. MIDAS, therefore, avoids giving any centrality to the literal interpretation.

11.3.1. Word-Senses Versus Conventional Metaphors

The distinction between a conventional metaphor and a fully lexicalized word-sense has become blurred in MIDAS. Many of the examples that have been presented here as conventional metaphors would have been treated as distinct unmotivated word-senses in most previous analyses. This section first considers the relationship between traditional word-senses and the representation of specific conventional metaphors. It then discusses the role that traditional word-senses play in the MIDAS approach.

Recall that, as discussed in Chapter 2, the distinct word-sense approach attempts to identify all the distinct uses of a given word and represent each as a separate fact in the lexicon. No attempt is made to relate these various meanings nor is any attempt made to account for the systematic use of related words. This approach succeeds in accomplishing two goals. The first is that it captures the intuition that these senses are conventional parts of the language, and should, therefore, be accounted for by specific knowledge. The second is that it provides an efficient basis for the interpretation of this non-literal conventional language.

One way to view the notion of a conventional metaphor is as an extension to the idea of a word-sense. Specific conventional metaphors can be viewed as structured word-senses, where the metaphoric motivation for the sense is represented directly as a part of the meaning. The conventionality of these uses is captured directly by the representation of the conventional metaphors in the same way that listing individual senses captures conventionality. The interpretation of these conventional metaphors is efficiently accomplished by mechanisms that are fundamentally the same as those used to apply knowledge about distinct word-senses. The metaphoric knowledge approach, therefore, captures conventionality and allows efficient interpretation, while at the same time capturing systematic metaphoric information that can not be captured by individual word-senses.

There are, of course, distinct word-senses that have no obvious systematic metaphoric basis. In the MIDAS approach, these senses are simply represented as distinct non-metaphoric concepts that are directly attached to the lexical item. During interpretation, each of these senses is considered along with all the other non-metaphoric conventional uses. MIDAS, therefore, handles non-metaphoric word-senses in basically the same way as in traditional approaches.

The key issue, however, is how to decide when to represent a use as a distinct sense as opposed to a metaphor. The bias in the MIDAS approach has been to treat each sense as a metaphor if there is any evidence that this use is systematically related to an existing metaphor. This decision must result from a fine-grained semantic analysis, along the lines of the analyses given in Chapter 3. This bias towards metaphoric motivations may lead to the representation of some senses as metaphors, when, in fact, they may actually be independent senses. However, as discussed above, this will not result in any significant problems for MIDAS. Sentences involving these uses can still be properly interpreted by the Metaphor Interpretation System. The only cost is in the space taken by the representation of the metaphor.

11.4. Learning

The primary objection to the knowledge-based approach to metaphor has always been that it doesn't handle "real" metaphors, ie., it can only deal with metaphors that are already a convention of the language. This thesis has shown that this need not be the case. The addition of explicit knowledge of the metaphors in the language radically alters the way that new metaphors are dealt with. It is no longer necessary to consider them as isolated anomalous inputs that have to be understood using special-purpose mechanisms. Rather, they are considered to be new, as yet unencountered, metaphors *that must coherently fit into* the system of existing known metaphors. The system of known metaphors, therefore, provides the basis for learning new ones.

When MIDAS learns a new metaphor there are two gains, one is immediate and one potential. The immediate gain is that the input example has been successfully interpreted and that this use is remembered in the form of a new metaphor. When this use is encountered again during future processing the newly learned metaphor can be used directly to interpret it correctly.

The potential gain has to do with future learning. In addition to serving as the conventional interpretation of metaphors of the same type, the newly learned metaphor can serve as a candidate metaphor for future learning. Consider the following examples from Chapters 5 and 10.

(1) Inflation is eating up our savings.
(2) Robbie's running ate up the distance.
(3) My car drinks gasoline.

When MIDAS originally learned the `Eat-Up-Reduce-Money` metaphor underlying (1), it was not anticipated that this metaphor would subsequently serve as the basis for an extension to (2) and (3). Each newly learned metaphor, therefore, has the potential to serve as the basis for a future extension.

11.5. Problems

There are four shortcomings in MIDAS's learning abilities that constitute areas for significant future research. The first is mainly a limitation in MIDAS, rather than a challenge to the overall approach. MIDAS's similarity and core-extension inferences clearly do not exhaust the ways in which known metaphors can be extended. In particular, these inferences do not address the issue of how to create new metaphors by combining existing ones.

Consider the metaphor underlying the phrase *communicate a disease*. This metaphor seems to be a combination of two existing metaphors: the Infection-As-Transfer metaphor, discussed earlier, and the Conduit metaphor (Reddy 1979). The Conduit metaphor allows communication to be viewed as a transfer, where ideas are objects that can be transferred and possessed. It seems that the Infection-As-Transfer and Communication-As-Transfer metaphors are combining to form an Infection-As-Communication metaphor. This combination of existing metaphors does not correspond to either a core or similarity extension. Although MIDAS cannot perform this kind of metaphor combination, it is not inconsistent with the overall model, since it involves the combination of known metaphors.

The second problem with MIDAS's learning model is more fundamental. Recall that a learning episode in MIDAS consists of two parts. First the meaning of the input metaphor is determined and then a new metaphor is created to represent this new use. While it seems reasonable to believe that a single instance of an unknown metaphor can be understood, it seems less likely that it would become an equal part of the understander's metaphoric knowledge, based upon a single instance. It seems more reasonable to believe that long-term knowledge of conventional metaphors is acquired and reinforced through repeated exposure. This kind of learning is not captured in MIDAS.

The next problem with MIDAS is that the issue of bootstrapping is not addressed. The learning mechanism relies on the existence of a set of already well-understood metaphors that form the basis for extension. The obvious question is where do the initial metaphors come from? There are two answers to this problem. The first answer is that there seems to be a built-in bias towards certain core metaphorical associations, based upon the way humans physically interact with the world (Gardner 1974, Lakoff 1986). A more practical answer, from the point of view of a computer system lacking this embodiment, is that these initial uses can be understood and remembered from contextual information using mechanisms similar to those suggested by (Granger 1977, Selfridge 1981, and Zernik 1987). Once these initial associations are formed, they can be extended using the techniques provided by MIDAS.

The final shortcoming in MIDAS has to do with target concept formation. MIDAS is concerned with the acquisition of metaphors that link well-understood source and target domains. It seems clear, however, that there are target concepts that are in large part learned almost exclusively through exposure to metaphors. Burstein (1986) partially addresses this issue in the context of students learning the semantics of variables in BASIC through exposure to a Container metaphor. The issue of concept formation through repeated exposure to systematic metaphors is a fertile one for future research.

There is a bright side to all of these problems. Once the step has been taken to deal with metaphor from a knowledge-based point of view, then the knowledge is there to be used for a wide variety of purposes. The development of MIDAS has shown that this knowledge can be used for both interpretation and learning tasks. Except for the bootstrapping issue, all of the above problems can be addressed by finding new and better ways to make use of this knowledge.

Appendix A

UNIX Examples

A.1 Introduction

This appendix demonstrates the pervasive nature of metaphor in the UNIX domain. The following UNIX examples will be demonstrated.

(1) How can I *get into* mail?
(2) How can I *get out* of emacs?

(3) How can I *kill* a file?

(4) You *have* write permission.
(5) You *need* write permission.
(6) Chmod *gives* you write permission.

The processing of Example (1) is an illustration of an extension by similarity, as described in Chapter 8. The system has knowledge of the `Enter-Lisp` metaphor. This metaphor will be extended, in this example, to cover entering the mail system by virtue of the hierarchical relationship between mail and lisp as processes.

```
> (do-sentence)
Interpreting sentence:

How can I get into mail?

Interpreting primal input.

(A Entering11 (↑ Entering)
   (agent12 (↑ agent) (A I12 (↑ I)))
```

```
    (patient12 (↑ patient) (A Mail11 (↑ Mail))))

Concreting input relations.

  Concreting  patient to entered.
  Concreting  agent to enterer.

Interpreting concreted input.

(A Entering11 (↑ Entering)
    (enterer11 (↑ enterer) (A I12 (↑ I)))
    (entered11 (↑ entered) (A Mail11 (↑ Mail))))


Failed interpretation: Entering11 as Entering.
Failed interpretation: Entering11 as Enter-Association.
Failed interpretation: Entering11 as Enter-Lisp.

No valid interpretations. Attempting to extend existing metaphor.
```

At this point, all the possible conventional interpretations of the primal input have been eliminated as potential readings. The input is now passed over to the Metaphor Extension System in an attempt to extend an existing metaphor to cover this new use and determine the intended meaning.

```
==========================================================
Entering Metaphor Extension System
==========================================================

Searching for related known metaphors.

Metaphors found: Enter-Lisp Enter-Association
```

The first step in the extension step is to collect all the relevant known metaphors that might be related to this new use. This initial search scans through all the metaphors directly attached to the input concept, and also at all the metaphors attached to concepts core related to the input concept. In this case, the system has knowledge of two metaphors that share the same source concept with the current use. The metaphors are ranked according to their conceptual distance from the input concepts.

```
Selecting metaphor Enter-Lisp to extend from.

(A Enter-Lisp  (↑ Container-Metaphor Metaphor-Schema)
   (enter-lisp-res enter-res → lisp-invoke-result)
   (lisp-enterer enterer → lisp-invoker)
```

```
   (entered-lisp entered → lisp-invoked)
   (enter-lisp-map Entering → Invoke-Lisp))
```

The metaphors whose target components are closest to the input concepts is selected as the candidate to try to extend. In this case, the `Enter-Lisp` metaphor is selected.

```
Attempting a similarity extension inference.

Extending similar metaphor Enter-Lisp with target concept Invoke-Lisp.

Abstracting Invoke-Lisp to ancestor concept Invoke-Comp-Process producing
abstract target meaning:

(A Invoke-Comp-Process7
   (↑ Invoke-Comp-Process)
   (comp-process-invoked7
    (↑ comp-process-invoked)
    (A Mail11 (↑ Mail)))
   (comp-process-invoker7
    (↑ comp-process-invoker) (A I12 (↑ I))))
```

The first step in the extension process is to abstract the candidate target to a concept high enough in the hierarchy to accept the input roles. In this case, the candidate target concept `Invoke-Lisp` is given the input roles of `I205` and `Mail23`. The first ancestor of `Invoke-Lisp` that can accept these inputs is the concept `Invoke-Comp-Process`. The input roles are then plugged into this more abstract concept to form a new instance of the category. This is shown as `Invoke-Comp-Process7`.

```
Concreting target concept Invoke-Comp-Process to Invoke-Mail producing
concreted meaning:

(A Invoke-Mail11 (↑ Invoke-Mail)
   (mail-invoker11 (↑ mail-invoker)
    (A I12 (↑ I)))
   (mail-invoked11 (↑ mail-invoked)
    (A Mail11 (↑ Mail))))
```

The abstracted concept is then concreted to the most specific concept that can accept the input roles. In this case, the concept `Invoke-Mail` is found that can cover the input.

This is the intended target meaning of the original metaphor.

```
Creating new metaphor:

 Mapping main source concept Entering to main target concept Invoke-Mail.
  Mapping source role enterer to target role mail-invoker.
  Mapping source role entered to target role mail-invoked.

(A Entering-Invoke-Mail (↑ Enter-Metaphor)
   (entered-mail-invoked-map entered → mail-invoked)
   (enterer-mail-invoker-map enterer → mail-invoker)
   (entering-invoke-mail-map Entering → Invoke-Mail))
```

Once an appropriate target meaning has been found, a new metaphor sense is created to reflect the fact that a new metaphor has been learned. This new metaphor is made a sibling of the metaphor that was used to obtain the meaning of the new use. In this case, the new metaphor is installed as a sibling of the `Enter-Lisp` metaphor. In the same fashion, the individual metaphor-maps that make up the new metaphor are made siblings of the corresponding maps in the candidate metaphor. For example, the `entered-mail-invoked-map` is installed as a child of the parent of the `entered-lisp` map that it corresponds to in the `Enter-Lisp` metaphor.

The names of the new metaphor and its maps are obtained by simply concatenating the names of the source and target concepts. In future processing, this metaphor can be applied directly by the conceptual analyzer and can be used by the Metaphor Extension System as a candidate for further extensions.

```
Final interpretation of input:

(A How-Q12 (↑ How-Q)
   (topic12 (↑ topic)
    (A Invoke-Mail11 (↑ Invoke-Mail)
       (mail-invoker11 (↑ mail-invoker)
     (A I12 (↑ I)))
       (mail-invoked11 (↑ mail-invoked)
     (A Mail11 (↑ Mail))))))
```

Once the final representation of the sentence has been obtained, it is passed along to the UC system. In this case, the input concept `Entering1` has been replaced by the concept `Invoke-Mail11` and assigned to the `topic` role of the concept `How-Q12` that represents the user's query.

```
Calling UC on input:

(A How-Q12 (↑ How-Q)
   (topic12 (↑ topic)
     (A Invoke-Mail11 (↑ Invoke-Mail)
        (mail-invoker11 (↑ mail-invoker)
      (A I12 (↑ I)))
        (mail-invoked11 (↑ mail-invoked)
      (A Mail11 (↑ Mail))))))

UC: You can enter mail by typing mail to shell.
```

In the following processing, Example (1) is processed again to show that the new metaphor has been learned. The new metaphor is applied directly, without MIDAS having to resort to its Metaphor Extension System.

```
> (do-sentence)
Interpreting sentence:

How can I get into mail?

Interpreting primal input.

(A Entering12 (↑ Entering)
   (agent13 (↑ agent) (A I13 (↑ I)))
   (patient13 (↑ patient) (A Mail12 (↑ Mail))))

Concreting input relations.

  Concreting  patient to entered.
  Concreting  agent to enterer.

Interpreting concreted input.

(A Entering12 (↑ Entering)
   (enterer12 (↑ enterer) (A I13 (↑ I)))
   (entered12 (↑ entered) (A Mail12 (↑ Mail))))


Failed interpretation: Entering12 as Entering.

Valid known metaphorical interpretation.
```

```
Applying conventional metaphor Entering-Invoke-Mail.

(A Entering-Invoke-Mail (↑ Enter-Metaphor)
   (entered-mail-invoked-map entered → mail-invoked)
   (enterer-mail-invoker-map enterer → mail-invoker)
   (entering-invoke-mail-map Entering → Invoke-Mail))

Mapping input concept Entering12 to concept Invoke-Mail12
  Mapping input role enterer12 with filler I13 to
       target role mail-invoker12
  Mapping input role entered12 with filler Mail12 to
       target role mail-invoked12
```

The new metaphor has been found and is being applied at this point.

```
Yielding interpretation:

(A Invoke-Mail12 (↑ Invoke-Mail)
   (mail-invoked12 (↑ mail-invoked)
    (A Mail12 (↑ Mail)))
   (mail-invoker12 (↑ mail-invoker)
    (A I13 (↑ I))))
```

The metaphor has been applied and the target meaning `Invoke-Mail` has instantiated with its roles filled in.

```
Failed interpretation: Entering12 as Enter-Association.
Failed interpretation: Entering12 as Enter-Lisp.
```

The system continues to check the other known interpretations. All the rest of the known interpretations fail as they did in the original example.

Since a valid conventional interpretation now exists based on the new metaphor, the MES is not called and the metaphorical interpretation is returned.

```
Final interpretation:

(A Invoke-Mail12 (↑ Invoke-Mail)
   (mail-invoked12 (↑ mail-invoked)
    (A Mail12 (↑ Mail)))
   (mail-invoker12 (↑ mail-invoker)
    (A I13 (↑ I))))

Final interpretation of input:
```

```
(A How-Q13 (↑ How-Q)
   (topic13 (↑ topic)
    (A Invoke-Mail12 (↑ Invoke-Mail)
       (mail-invoked12 (↑ mail-invoked)
     (A Mail12 (↑ Mail)))
       (mail-invoker12 (↑ mail-invoker)
     (A I13 (↑ I))))))
```

Once again, this final interpretation of the original question is sent to the UC system for answering.

```
Calling UC on input:

(A How-Q13 (↑ How-Q)
   (topic13 (↑ topic)
    (A Invoke-Mail12 (↑ Invoke-Mail)
       (mail-invoked12 (↑ mail-invoked)
     (A Mail12 (↑ Mail)))
       (mail-invoker12 (↑ mail-invoker)
     (A I13 (↑ I))))))

UC: You can enter mail by typing mail to the shell.
```

The previous example was an example of a normal similarity extension inference. MIDAS handled a new metaphorical use of `Entering` based upon a previously understood entering metaphor. The following example illustrates the system's ability to combine evidence in the form of combined core extension and similarity extension inferences. In the following processing of Example (6), MIDAS learns the meaning of 'exit emacs' from the known metaphor `Enter-Lisp`. This metaphor is connected to the input example by the core relationship from `Entering` to `Exiting` and the similarity relationship between `Lisp` and `Emacs`.

```
> (do-sentence)
Interpreting sentence:

How can I get out of emacs?

Interpreting primal input.

(A Exiting98 (↑ Exiting)
   (agent603 (↑ agent) (A I206 (↑ I)))
   (patient568 (↑ patient)
```

```
    (A Emacs45 (↑ Emacs))))

Concreting input relations.

  Concreting  patient to exited.
  Concreting  agent to exiter.

Interpreting concreted input.

(A Exiting98 (↑ Exiting)
   (exiter59 (↑ exiter) (A I206 (↑ I)))
   (exited59 (↑ exited) (A Emacs45 (↑ Emacs))))

Failed interpretation: Exiting98 as Exiting.

No valid interpretations. Attempting to extend existing metaphor.

==========================================================
Entering Metaphor Extension System
==========================================================


Searching for related known metaphors.

Metaphors found: Enter-Lisp Enter-Association

Selecting metaphor Enter-Lisp to extend from.
```

Once again the system selects from among the existing known metaphors. The `Enter-Lisp` metaphor is again selected because it is closest to the new example. Note that the `Entering-Invoke-Mail` metaphor, produced in the previous example, was removed from the knowledge base before this example was run. Had it been left in then it would have been selected and processed as a normal core related extension.

```
(A Enter-Lisp  (↑ Container-Metaphor Metaphor-Schema)
   (enter-lisp-res enter-res → lisp-invoke-result)
   (lisp-enterer enterer → lisp-invoker)
   (entered-lisp entered → lisp-invoked)
   (enter-lisp-map Entering → Invoke-Lisp))

Attempting a similarity core-related metaphorical extension.
```

The system has determined that, based on the shape of the connection from the input example to the `Enter-Lisp` metaphor, that this is a core-related similarity inference. Ie.

an example that combines a core and similarity situation.

```
Extending similar core-related metaphor Enter-Lisp
with target concept Invoke-Lisp.

This is an intermediate extension inference.

Applying source path:

 Exiting → exit-pre → Enclosed-State → enter-res* → Entering

to target concept Invoke-Lisp yields target connection.

 Invoke-Lisp → lisp-invoke-result → Lisp-Active → lisp-uninvoke-pre*
→ Uninvoke-Lisp


Applying source path yields target concept Uninvoke-Lisp.
```

The next step in applying a similar core related metaphor is to find the core related target concept via the application of source path to the target concept of the candidate-metaphor. At this point, the system has applied the source core path relation to the target domain and located the target concept `Uninvoke-Lisp`. This concept must be processed in the same fashion as a normal similarity extension.

```
Abstracting Uninvoke-Lisp to ancestor concept Uninvoke-Comp-Process
producing abstract target meaning:

(A Uninvoke-Comp-Process53
   (↑ Uninvoke-Comp-Process)
   (comp-process-uninvoked53
    (↑ comp-process-uninvoked)
    (A Emacs45 (↑ Emacs)))
   (comp-process-uninvoker53
    (↑ comp-process-uninvoker) (A I206 (↑ I))))

Concreting Uninvoke-Comp-Process53 to concept Uninvoke-Emacs.

Yielding concept:

(A Uninvoke-Emacs39 (↑ Uninvoke-Emacs)
   (emacs-uninvoker39 (↑ emacs-uninvoker)
    (A I206 (↑ I)))
   (emacs-uninvoked39 (↑ emacs-uninvoked)
    (A Emacs45 (↑ Emacs))))
```

The concept `Uninvoke-Lisp` is abstracted to the concept `Univoke-Comp-Process` since this is the first ancestor of the `Uninvoke-lisp` concept whose constraints can accept the input concept `Emacs`. At this point, the input roles are assigned and the concept is concreted to the intended meaning of `Univoke-Emacs`.

```
Creating new metaphor:

 Mapping main source concept Exiting to main target concept Uninvoke-Emacs.
  Mapping source role exiter to target role emacs-uninvoker.
  Mapping source role exited to target role emacs-uninvoked.

(A Exiting-Uninvoke-Mail (↑ Exiting-Metaphor)
   (emacs-uninvoked-map exited → emacs-uninvoked)
   (emacs-uninvoker-map exiter → emacs-uninvoker)
   (exiting-uninvoke-emacs-map Exiting → Uninvoke-Emacs))
```

As in the previous learning examples, the new example leads to the creation of a new metaphor.

```
Final interpretation of input:

(A How-Q210 (↑ How-Q)
   (topic209 (↑ topic)
    (A Uninvoke-Emacs39 (↑ Uninvoke-Emacs)
       (emacs-uninvoker39 (↑ emacs-uninvoker)
     (A I206 (↑ I)))
       (emacs-uninvoked39 (↑ emacs-uninvoked)
     (A Emacs45 (↑ Emacs))))))

Calling UC on input:

(A How-Q210 (↑ How-Q)
   (topic209 (↑ topic)
    (A Uninvoke-Emacs39 (↑ Uninvoke-Emacs)
       (emacs-uninvoker39 (↑ emacs-uninvoker)
     (A I206 (↑ I)))
       (emacs-uninvoked39 (↑ emacs-uninvoked)
     (A Emacs45 (↑ Emacs))))))

UC: You can exit emacs by typing ^x^c.
```

The processing of Example (11) illustrates a major issue in the area of conventional metaphor having to do with lexicalization and learning. A basic assumption of the system is that the user will be using domain metaphors in a normal or conventional fashion.

Metaphors that are extremely novel or incoherent will not be understood by the system because there will be no known metaphor that is sufficiently close to them to be easily understood. There are, however, coherent metaphors that are simply not a part of conventional usage. Consider the following example.

(7) How can I kill a file?

The use of the word *kill* to mean delete is a part of the UNIX lexicon when applied to lines of text and characters. One can 'kill a line' or 'use the kill character'. This use of *kill* does not however conventionally extend to the deletion of files. It is not normal UNIX language to refer to 'deleting a file' as killing. The system can however come to a correct interpretation of this use by a similarity extension from the 'kill a line' use to mean delete. The following trace illustrates this processing.

```
> (do-sentence)
Interpreting sentence:

How can I kill a file?

Interpreting primal input.

(A Killing145 (↑ Killing)
   (agent601 (↑ agent) (A I204 (↑ I)))
   (patient566 (↑ patient)
    (A Text-File22 (↑ Text-File))))

Concreting input relations.

  Concreting  patient to kill-victim.
  Concreting  agent to killer.

Interpreting concreted input.

(A Killing145 (↑ Killing)
   (killer90 (↑ killer) (A I204 (↑ I)))
   (kill-victim90 (↑ kill-victim)
    (A Text-File22 (↑ Text-File))))

Failed interpretation: Killing145 as Killing.
Failed interpretation: Killing145 as Kill-Delete-Line.
Failed interpretation: Killing145 as Kill-Sports-Defeat.
Failed interpretation: Killing145 as Kill-Conversation.

No valid interpretations. Attempting to extend existing metaphor.
```

```
==============================================================
Entering Metaphor Extension System
==============================================================

Searching for related known metaphors.

Metaphors found: Kill-Delete-Line Kill-Conversation Kill-Sports-Defeat

Selecting metaphor Kill-Delete-Line to extend from.

(A Kill-Delete-Line  (↑ Kill-Metaphor Metaphor-Schema)
   (killed-deleted kill-victim → line-deleted)
   (killer-deleter killer → line-deleter)
   (kill-delete Killing → Delete-Line))

Attempting a similarity extension inference.

Extending similar metaphor Kill-Delete-Line with target
concept Delete-Line.

Abstracting Delete-Line to ancestor concept Deleting producing
abstract target meaning:

(A Deleting22 (↑ Deleting)
   (deleted22 (↑ deleted)
    (A Text-File22 (↑ Text-File)))
   (deleter22 (↑ deleter) (A I204 (↑ I))))

Concreting target concept Deleting to Delete-File producing
concreted meaning:

(A Delete-File20 (↑ Delete-File)
   (file-deleter20 (↑ file-deleter)
    (A I204 (↑ I)))
   (file-deleted20 (↑ file-deleted)
    (A Text-File22 (↑ Text-File))))

Creating new metaphor:

 Mapping main source concept Killing to main target concept Delete-File.
  Mapping source role kill-victim to target role file-deleted.
  Mapping source role killer to target role file-deleter.

(A Killing-Delete-File (↑ Kill-Metaphor)
   (kill-victim-file-deleted-map kill-victim → file-deleted)
   (killer-file-deleter-map killer → file-deleter)
   (killing-delete-file-map Killing → Delete-File))
```

At this point, the system has created a new metaphor, `Killing-Delete-File` to represent the use of kill a file to mean delete. This is instantiated as a sibling `Kill-Delete-Line` metaphor in the hierarchy.

```
Final interpretation of input:

(A How-Q208 (↑ How-Q)
   (topic207 (↑ topic)
    (A Delete-File20 (↑ Delete-File)
      (file-deleter20 (↑ file-deleter)
     (A I204 (↑ I)))
      (file-deleted20 (↑ file-deleted)
     (A Text-File22 (↑ Text-File))))))

Calling UC on input:

(A How-Q208 (↑ How-Q)
   (topic207 (↑ topic)
    (A Delete-File20 (↑ Delete-File)
      (file-deleter20 (↑ file-deleter)
     (A I204 (↑ I)))
      (file-deleted20 (↑ file-deleted)
     (A Text-File22 (↑ Text-File))))))
```

The UC generator again uses the user's metaphorical language when generating the answer to this query.

```
UC: You can kill a file by typing rm filename.
```

The main issue here is that while the system can and should understand this use of *kill* to mean delete, it should not become a part of its normal lexicon. In particular, the system should not normally describe the workings of the result of the 'rm' command as a killing. The problem is that a single-instance learning system has no way of distinguishing a coherent metaphor that simply is not conventional from one that is both coherent and conventional.

The following examples from Jacobs (1985) illustrate a metaphorical UNIX use of the concepts giving and having.

```
> (do-sentence)
Interpreting sentence:

You have write permission.

Interpreting primal input.

(A Having51 (↑ Having)
   (agent598 (↑ agent) (A You39 (↑ You)))
   (patient563 (↑ patient)
    (A Write-Permission42 (↑ Write-Permission))))

Concreting input relations.

  Concreting  patient to had.
  Concreting  agent to haver.

Interpreting concreted input.

(A Having51 (↑ Having)
   (haver42 (↑ haver) (A You39 (↑ You)))
   (had42 (↑ had)
    (A Write-Permission42 (↑ Write-Permission))))

Failed interpretation: Having51 as Having.
Failed interpretation: Having51 as Have-Idea.
Failed interpretation: Having51 as Have-Cold.

Valid known metaphorical interpretation.

Applying conventional metaphor Have-Permission.

(A Have-Permission  (↑ Have-State Metaphor-Schema)
   (had-permission had → permitted)
   (haver-permitted haver → permittee)
   (having-permitted Having → Permitted-State))

Mapping input concept Having51 to concept Write-Permitted-State14
  Mapping input role had42 with filler Write-Permission42 to
        target role write-action-permitted14
  Mapping input role haver42 with filler You39 to
        target role write-permitted12

Yielding interpretation:
```

```
(A Write-Permitted-State14 (↑ Write-Permitted-State)
   (write-permitted12 (↑ write-permitted)
    (A You39 (↑ You)))
   (write-action-permitted14 (↑ write-action-permitted)
    (A Write-Permission42 (↑ Write-Permission))))


Final interpretation of input:

(A Write-Permitted-State14 (↑ Write-Permitted-State)
   (write-permitted12 (↑ write-permitted)
    (A You39 (↑ You)))
   (write-action-permitted14 (↑ write-action-permitted)
    (A Write-Permission42 (↑ Write-Permission))))
```

The following two examples illustrate direct extension inferences from the core `have-permission` metaphor.

```
> (do-sentence)
Interpreting sentence:

You need write permission.

Interpreting primal input.

(A Needing30 (↑ Needing)
   (agent599 (↑ agent) (A You40 (↑ You)))
   (patient564 (↑ patient)
    (A Write-Permission43 (↑ Write-Permission))))

Concreting input relations.

  Concreting  patient to needed.
  Concreting  agent to needer.

Interpreting concreted input.

(A Needing30 (↑ Needing)
   (needer26 (↑ needer) (A You40 (↑ You)))
   (needed26 (↑ needed)
    (A Write-Permission43 (↑ Write-Permission))))

Failed interpretation: Needing30 as Needing.
```

```
No valid interpretations. Attempting to extend existing metaphor.

===========================================================
Entering Metaphor Extension System
===========================================================


Searching for related known metaphors.

Metaphors found: Have-Permission Have-Idea Have-Cold Have-State
 Get-Grade Give-Flu

Selecting metaphor Have-Permission to extend from.

(A Have-Permission  (↑ Have-State Metaphor-Schema)
   (had-permission had → permitted)
   (haver-permitted haver → permittee)
   (having-permitted Having → Permitted-State))

Attempting a core-related metaphorical extension.

Extending similar core-related metaphor Have-Permission
with target concept Permitted-State.

This is a direct extension inference.


Applying source path:

Needing → needed-state → Having

to target concept Permitted-State yields target connection.

Permitted-State → req-permission* → Required-Permission

Applying source path yields target concept Required-Permission.

(A Required-Permission4
   (↑ Required-Permission)
   (req-permittee4 (↑ req-permittee)
    (A You40 (↑ You)))
   (req-permitted-of4 (↑ req-permitted-of)
    (A Write-Permission43 (↑ Write-Permission))))

Creating new metaphor:

 Mapping main source concept Needing to main target concept
```

```
Required-Permission.
  Mapping source role needer to target role req-permittee.
  Mapping source role needed to target role req-permitted-of.

(A Needing-Required-Permission  (↑ Metaphor-Schema)
   (needed-req-permitted-of-map needed → req-permitted-of)
   (needer-req-permittee-map needer → req-permittee)
   (needing-required-permission-map Needing → Required-Permission))

Final interpretation of input:

(A Required-Permission4
   (↑ Required-Permission)
   (req-permittee4 (↑ req-permittee)
    (A You40 (↑ You)))
   (req-permitted-of4 (↑ req-permitted-of)
    (A Write-Permission43 (↑ Write-Permission))))
```

```
> (do-sentence)
Interpreting sentence:

Chmod gives you write permission.

Interpreting primal input.

(A Giving95 (↑ Giving)
   (agent600 (↑ agent) (A Chmod6 (↑ Chmod)))
   (patient565 (↑ patient) (A You41 (↑ You)))
   (object125 (↑ object)
    (A Write-Permission44 (↑ Write-Permission))))

Concreting input relations.

  Concreting  object to given.
  Concreting  patient to givee.
  Concreting  agent to giver.

Interpreting concreted input.

(A Giving95 (↑ Giving)
   (giver54 (↑ giver) (A Chmod6 (↑ Chmod)))
   (givee54 (↑ givee) (A You41 (↑ You)))
   (given55 (↑ given)
```

```
    (A Write-Permission44 (↑ Write-Permission))))

Failed interpretation: Giving95 as Giving.
Failed interpretation: Giving95 as Give-Flu.

No valid interpretations. Attempting to extend existing metaphor.

==========================================================
Entering Metaphor Extension System
==========================================================


Searching for related known metaphors.

Metaphors found: Have-Permission Needing-Required-Permission Have-Idea
 Have-Cold Have-State Get-Grade Give-Flu

Selecting metaphor Have-Permission to extend from.

(A Have-Permission  (↑ Have-State Metaphor-Schema)
   (had-permission had → permitted)
   (haver-permitted haver → permittee)
   (having-permitted Having → Permitted-State))

Attempting a core-related metaphorical extension.


Extending similar core-related metaphor Have-Permission
with target concept Permitted-State.

This is a direct extension inference.


Applying source path:

Giving → give-result → Having

to target concept Permitted-State yields target connection.

Permitted-State → permit-result* → Permit-Action


Applying source path yields target concept Permit-Action.

(A Permit-Action2 (↑ Permit-Action)
   (permit-permitter2 (↑ permit-permitter)
    (A Chmod6 (↑ Chmod)))
```

```
   (permit-permittee2 (↑ permit-permittee)
    (A You41 (↑ You)))
   (permission-permitted2
    (↑ permission-permitted)
    (A Write-Permission44 (↑ Write-Permission))))
```

Creating new metaphor:

```
 Mapping main source concept Giving to main target concept Permit-Action.
  Mapping source role giver to target role permit-permitter.
  Mapping source role givee to target role permit-permittee.
  Mapping source role given to target role permission-permitted.
```

```
(A Giving-Permit-Action (↑ Metaphor-Schema)
  (given-permission-permitted-map given → permission-permitted)
   (givee-permit-permittee-map givee → permit-permittee)
   (giver-permit-permitter-map giver → permit-permitter)
   (giving-permit-action-map Giving → Permit-Action))
```

Final interpretation of input:

```
(A Permit-Action2 (↑ Permit-Action)
   (permit-permitter2 (↑ permit-permitter)
    (A Chmod6 (↑ Chmod)))
   (permit-permittee2 (↑ permit-permittee)
    (A You41 (↑ You)))
   (permission-permitted2
    (↑ permission-permitted)
    (A Write-Permission44 (↑ Write-Permission))))
```

Appendix B

Implementation Status

B.1 Introduction

This appendix presents some of the implementation details of MIDAS and KODIAK. These programs were written in Common Lisp on a Texas Instruments Explorer I lisp machine.

B.2 KODIAK

The major difference between previous implementations of KODIAK and the one used here is that primitive links are not explicitly represented as objects in the knowledge-base. In previous implementations, the basic data types were *absolute*, *relation*, *aspectual*, and *link*. All were defined using the defstruct facility. Each link object possessed a *From* and *To* field that specified the domain and range of the link. Absolutes, relations, and aspectuals contained a field that listed the incoming and outgoing links attached to that concept.

There were two major problems with this technique. The resulting knowledge-base was much larger than necessary because of the proliferation of links, as a result reloading the knowledge became very time-consuming. The second problem was processing speed. In order to traverse a single domain relation from the domain to the range, four links and five concepts had to be traversed. Operations that should have been primitive and fast were taking too much time and space.

The implementation used in the current version implicitly represents links as fields in normal concepts. Absolutes, relations, and aspectuals are provided with the following fields: name, parents, ancestors, children, descendents, relations, equates, and differs. This design resulted in a much more compact knowledge-base that could be traversed and loaded much more quickly.

The function called most frequently by MIDAS and KODIAK is called `Ancestor-`

`Or-Eq-P`. This function takes two concepts and checks if the first concept ultimately dominates the second concept. The original implementation of this function involved searching up through the hierarchy to find ancestors. This was necessitated by the fact that the representation only stored the immediate parents of each concept. This function quickly became a major bottleneck. `Ancestor-Or-Eq-P` was subsequently augmented with a caching mechanism. Whenever it is first called on a concept, the full list of ancestors is computed and then stored in the ancestor field of that concept. Subsequent calls to `Ancestor-Or-Eq-P` merely look at this field rather than searching the hierarchy. This effectively led to a five-fold speed-up in MIDAS's overall performance.

The version of KODIAK used by MIDAS was also augmented with a set of tools to manipulate and create metaphors. The most useful user level function is called `Defmetaphor`. An example of this is shown below in Figure 1.

```
(defmetaphor give-cold
  (give-infect-map   giving → cold-infect)
  (giver-infector    giver  → cold-infector)
  (givee-infected    givee  → cold-victim)
  (given-infection   given  → infected-cold)
  (give-infect-res   give-result → cold-inf-res))
```

Figure 1: Creating a Metaphor

`Defmetaphor` is used to create new metaphor-senses along with all the attendant metaphor-maps. In the above example, the `Give-Cold` metaphor is created with all its source target associations. The user merely specifies the name of the metaphor and its associations. The concretion mechanism is used to automatically classify the metaphor-sense and all the new metaphor-maps.

B.3 Statistics

The size of MIDAS's knowledge-based is summarized in Figure 1. Since MIDAS has a learning component, the size of the knowledge-base is difficult to state definitively. The initial knowledge-base contained 13 metaphor-senses that were hand-coded. During the course of processing the examples discussed in this thesis, an additional 23 metaphor-senses were acquired. This, of course, is not intended to represent the full set of metaphors derivable from the base set.

The interpretation of sentences that contain conventional metaphors known to MIDAS averaged 1.4 seconds of real time (2.6 seconds for UC examples). On average, less than .1 seconds were spent in the initial parse phase, the rest being spent in semantic

Concept Type	Number
Absolutes	243
Relations	415
Aspectuals	830
Original KB	
Metaphor-Senses	13
Metaphor-Maps	35
Final KB	
Metaphor-Senses	36
Metaphor-Maps	94

Size of Knowledge-Base

interpretation and UC processing. Examples involving the interpretation and acquisition of new metaphors averaged 7.8 seconds.

An important question to ask of any learning system is what effect the addition of the new knowledge has on the overall system performance. For MIDAS, at least as far as time is concerned, there are two metrics. The first is the speed-up experienced by MIDAS when newly learned metaphors are again encountered. There was, on average, a three-fold speedup in the subsequent processing of examples involving newly learned metaphors. The overhead costs of adding a new metaphor incurred during the processing of metaphors not related to the new one were negligible. In other words, the addition of a new metaphor does not significantly effect the time taken to process sentences where it is not relevant. These times are all for real time running compiled Common Lisp on a TI Explorer I, running Release 3.2.

References

Alterman, R., Adaptive Planning. *Cognitive Science 12*. pp. 393-421, 1988.

Anderson, J. R., *The Architecture of Cognition*. Harvard University Press, Cambridge, MA, 1983.

Berwick, R. *The Acquisition of Syntactic Knowledge*. MIT Press, Cambridge, MA, 1985.

Black, M., Metaphor. In *Proceedings of the Aristotlean Society*, pp. 273-294, 1954.

Black, M., *Models and Metaphors*. Cornell University Press, Ithaca, NY, 1962.

Bobrow, D. and Winograd, T., An Overview of KRL: A Knowledge Representation Language. *Cognitive Science 1*, pp. 346-370, 1977.

Brachman, R. J., and Schmolze, J. G., An Overview of the KL-ONE Knowledge Representation System. *Cognitive Science 9*, pp. 171-216, 1985.

Braverman, M.S., and Russell, S.J., Boundaries of Operationality. In *Proceedings of the Fifth International Conference on Machine Learning*. pp. 221-234, Ann Arbor, MI, 1988.

Brugman, C., The Story of Over. Unpublished Masters Thesis. University of California, Berkeley. 1981.

Burstein, M.H., Concept Formation by Incremental Analogical Reasoning and Debugging. In *Machine Learning: Volume II*. Michalski, R.S., Carbonell, J.G., Mitchell, T.M., (eds), Morgan Kaufmann, Los Altos, CA. 1986.

Carbonell, J.G., Metaphor: A Key to Extensible Semantic Analysis. *Proceedings of the 18th Meeting of the Association for Computational Linguistics*, 1980.

Carbonell, J. G., Invariance Hierarchies in Metaphor Interpretation. *Proceedings of the Third Meeting of the Cognitive Science Society*. Cognitive Science Society, pp. 292-295, August 1981.

DeJong, G.F. and Waltz, D.L., Understanding Novel Language. *Computers and Mathematics with Applications 9*. 1983.

Falkenhainer, B., Forbus, K. D., and Gentner, D. The Structure Mapping Engine. In *Proceedings of AAAI-86*. Philadelphia, PA. 1986.

Fass, D. and Wilks, Y., Preference Semantics, Ill-Formedness and Metaphor. In *American Journal of Computational Linguistics 9*, pp. 178-187, 1983.

Fass, D., Collative Semantics: A Description of the Meta5 Program. Memoranda in Cognitive and Computer Science. Report No. MCCS-86-23, New Mexico State University, 1986.

Fass, D., *Collative Semantics: A Semantics for Natural Language Processing*. PhD Thesis, Report No. MCCS-88-118, Computing Research Laboratory, New Mexico State University, NM, 1988.

Fillmore, C. Frame Semantics. In *Linguistics in the Morning Calm* Linguistics Society of Korea, Hanshin, Korea, 1982.

Gardner, H., Metaphors and Modalities: How Children Project Polar Adjectives onto Diverse Domains. *Child Development, 45*, pp. 84-91. 1974.

Gentner, D., Some Interesting Differences Between Verbs and Nouns. *Cognition and Brain Theory, 4*, pp. 161-178, 1981.

Gentner, D., Structure Mapping: A Theoretical Framework for Analogy. *Cognitive Science 7* pp. 155-170. 1983.

Gentner, D. and Stuart, P., Metaphor as Structure Mapping: What Develops. BBN Technical Report No. 5479, Cambridge, MA 1983.

Gentner, D., Falkenhainer, B., Skorstad, J., Viewing Metaphor as Analogy. In *Analogical Reasoning*. Helman, D.H., (ed), Kluwer Academic Publishers, London, 1988.

Gentner.D. and France, I.M., The Verb Mutability Effect: Studies of the Combinatorial Semantics of Nouns and Verbs. In *Lexical Ambiguity Resolution*. Small, S., Cottrell, G., Tanenhaus, M., (eds), Morgan Kaufmann Publishers, San Mateo, CA, 1988.

Granger, R.H., FOUL-UP: A Program That Figures Out Meanings of Words from Context. *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*. Cambridge, MA, 1977.

Greiner, R. *Learning by Understanding Analogies*. PhD Thesis, Stanford University, Report No. STAN-CS-85-1071, 1985.

Greiner, R., Abstraction-Based Analogical Inference. In *Analogical Reasoning*. Helman, D.H., (ed), Kluwer Academic Publishers, London, 1988.

Helman, D.H., (ed), *Analogical Reasoning*, Kluwer Academic Publishers, Boston, 1988.

Hammond, K., *Case-Based Planning: An Integrated Theory of Planning, Learning, and Memory*. PhD. Thesis, Yale University, Department of Computer Science Report No. 488, 1986.

Herskovits, A., *Language and Spatial Cognition*. Cambridge University Press, Cambridge, 1986.

Hirst, G., *Semantic Interpretation and the Resolution of Ambiguity*. Cambridge University Press, Cambridge. 1987.

Hobbs, J., Metaphor, Metaphor Schemata, and Selective Inferencing. SRI Technical Note 204, December 1979.

Indurkhya, B., Approximate Semantic Transference: A Computational Theory of Metaphors and Analogy. *Cognitive Science , 11*. pp. 445-480, 1987.

Jackendoff, R., *Semantics and Cognition*. MIT Press, Cambridge, MA, 1983.

Jacobs, P.,S., *A Knowledge-Based Approach to Language Production*. PhD. Thesis. University of California, Berkeley, Report No. UCB/CSD 86/254, August 1985.

Jacobs, P.,S., Knowledge-Intensive Natural Language Generation, *Artificial Intelligence, 33*. pp. 325-378, 1987.

Johnson, M. (Ed)., *Philosophical Perspectives on Metaphor*. University of Minnesota Press, Minneapolis, MN., 1981.

Katz, J. and Fodor, J.A., The Structure of a Semantic Theory. *Language, 39*, pp. 170-230, 1963.

Lakoff, G. and Johnson, M., *Metaphors We Live By*. University of Chicago, 1980.

Lakoff, G., *Women, Fire and Dangerous Things*. University of Chicago, Chicago, IL, 1986.

Lakoff, G. and Turner, M., *More Than Cool Reason*. University of Chicago Press, Chicago, IL, In Press.

Langacker, R. *Foundations of Cognitive Grammar*. Stanford University Press, Stanford, CA, 1987.

Lindner, S., *A Lexico-Semantic Analysis of Verb-Particle Constructions with "Up"*. PhD. Thesis. University of California, San Diego, 1981.

Luria, M., *Knowledge Intensive Planning*. PhD. Thesis. University of California, Berkeley, Report No. UCB/CSD 88/433, 1988.

Martin, J., Knowledge Acquisition through Natural Language Dialogue. In *Proceedings of the 2nd Conference on Artificial Intelligence Applications*. Miami, Florida, December 1985.

Martin, J., Views From a Kill. In *Proceedings of the 8th National Conference of the Cognitive Science Society*. Amherst, MA, August 1986.

Martin, J., Representing and Acquiring Knowledge About Metaphors. *Proceedings of the 3rd Workshop on Theoretical Issues in Conceptual Information Processing*. Philadelphia, PA, 1986.

Martin, J., Understanding New Metaphors. *Proceedings of IJCAI-87*. Milan, Italy, 1987.

Martin, J., Representing Regularities in the Metaphoric Lexicon. *Proceedings of the 12th International Conference on Computational Linguistics*. Budapest, Hungary, 1988.

Norvig, P., Categorizing the Senses of Take. *Proceedings of the 3rd Workshop on Theoretical Issues in Conceptual Information Processing*. Philadelphia, PA, 1986.

Norvig, P., *A Unified Theory of Inference For Text Understanding*. PhD. Thesis, University of California, Berkeley, Report No. UCB/CSD 87/339, 1987.

Ortony, A. (Ed)., *Metaphor and Thought* Cambridge University Press, 1979.

Prieditis, A., (ed), *Analogica: Processdings of the First Workshop on Analogical Reasoning*, Morgan Kaufmann, Los Altos, Ca, 1987.

Pinker, S., *Language Learnability and Language Development*. Harvard University Press, Cambridge, MA, 1984.

Reddy, M., The Conduit Metaphor. In *Metaphor and Thought.* Ortony, A. (Ed). pp. 284-324, Cambridge University Press, Cambridge, 1979.

Richards, I.,A., *The Philosophy of Rhetoric.* Oxford University Press, Oxford, 1936.

Riesbeck, C., Realistic Language Comprehension. In, *Strategies for Natural Language Processing.* Lehnert, W. and Ringle, M. (Eds), Lawrence Erlbaum Associates, Hillsdale, NJ, 1982.

Russell, S.W., Computer Understanding of Metaphorically Used Verbs. *American Journal of Computational Linguistics.* Microfiche 44, 1976.

Schank, R. and Abelson, R., *Scripts, Plans, Goals and Understanding* Lawrence Erlbaum Associates, Hillsdale, NJ, 1977.

Selfridge, M., A Computer Model of Child Language Acquisition. In *Proceedings of the Seventh International Joint Conference on Artificial Intelligence.* pp. 92-96, Vancouver, Canada, 1981.

Small, S. and Rieger, C., Parsing and Comprehending with Word Experts. In *Strategies for Natural Language Processing.* Lehnert, W. and Ringle, M. (Eds), Lawrence Erlbaum Associates, Hillsdale, NJ, 1982.

Weiner, E. J., A Knowledge Representation Approach to Understanding Metaphors. *Computational Linguistics 10.* 1984.

Wilensky, R., KODIAK: A Knowledge Representation Language. In *Proceedings of the 6th National Conference of the Cognitive Science Society.* Boulder, CO, June 1984.

Wilensky, R., Arens, Y. and Chin, D., Talking to Unix in English: An overview of UC. In *Comm. ACM 27.* pp. 574-593, June 1984.

Wilensky, R., Some Problems and Proposals for Knowledge Representation. University of California, Berkeley, Computer Science Division Report No. UCB/CSD 86/294, May 1986.

Wilensky, R. et al, UC: A Progress Report. University of California, Berkeley, Computer Science Division Report No. UCB/CSD 87/303, 1986.

Wilks, Y., Preference Semantics. In *The Formal Semantics of Natural Language* Keenan (ed), Cambridge, 1975.

Wilks, Y., Making Preferences More Active. *Artificial Intelligence 11.* 1978.

Winston P., Learning by Creating Transfer Frames. *Artificial Intelligence 10.* 1978.

Winston P., Learning and Reasoning by Analogy. *Comm. ACM 23.* pp. 689-703, December 1980.

Zernik, U., *Strategies in Language Acquisition: Learning Phrases from Examples in Context.* PhD. Thesis, University of California, Los Angeles, Report No. UCLA-AI-87-1, 1987.